

Project I



Exactly 1 meeting between your group and a TA

Resolve your post if you found someone!

The screenshot shows a forum thread titled "Keys for Weak Entities" by "General Jiaxiang Liu". The thread has 17h activity and 4 replies. The first reply is from "Victor Odorn" (2d) saying "Let's gooooo". The second reply is from "Yizhang Huang" (5d) with the status "Unresolved", stating: "Hi everyone! My name is Yizhang(Sam) Huang and I'm currently a graduate student majoring in statistics. I had some experience in SQL but not a lot of experience in app development. Here is my email yh3674@columbia.edu. Let me know if you wanna team up!". The third reply is from "Amanda Li" (5d) with the status "Unresolved", saying "Hi everyone! My name is Amanda. I did my undergrad at UC Berkeley and am currently in the Bridge CS program at Columbia. I have some experience with SQL for databases from previous classes and jobs. I hope to conduct a project with interesting real-life applications. Please feel free to message me if you are interested in teaming up :) My email is aj231@columbia.edu. You can also text me via 309 536 9520. Thanks!". The fourth reply is from "Raymibek Kokteubay" (5d) with the status "Unresolved", saying "Just sent an email!". The fifth reply is from "Ulemj Munkhtur" (6d) with the status "Unresolved", saying "Hi everyone!". The sidebar on the left shows other threads and categories like "RSES", "11 002 F24", "hw1 Posted", "Finding Project Teammates Megathread", "test post", "hw1 specialization hierarchy IDs", "HWO Submission", and "This Week".

Office hours

Jordyn's OH changing

- Weds this week was to add more OH before HWI due date

Chalk Board Examples

Verbs vs Nouns, or Action vs Information

Weak Entities

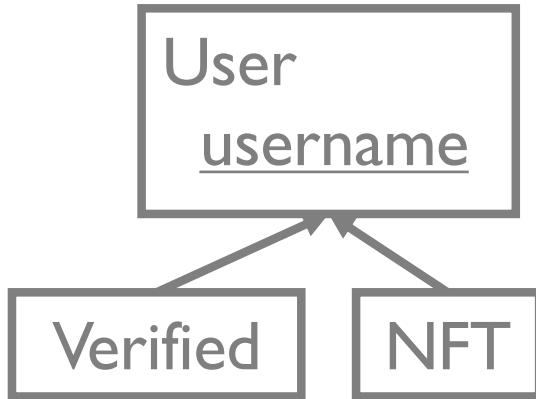
Relationships vs Relationship Set

QUIZ!

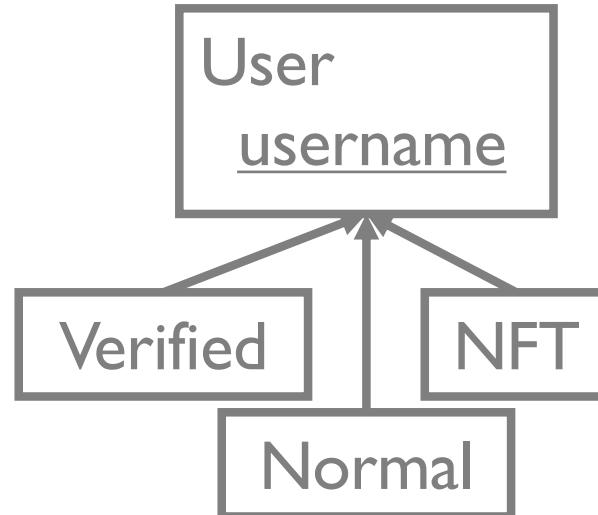


<http://w4lll.github.io/quiz.html>

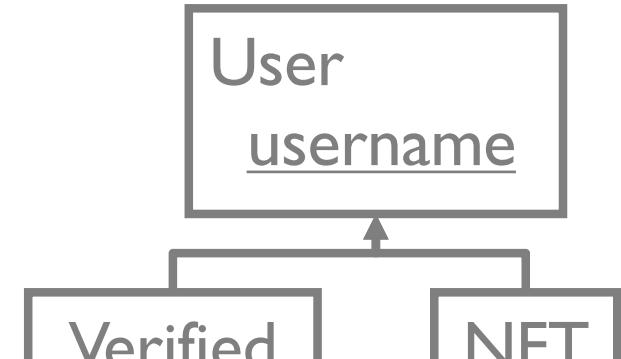
A twitter **user** is identified by username, and can be a **verified user**, an **NFT user**, a **normal** user, or any combination.



A



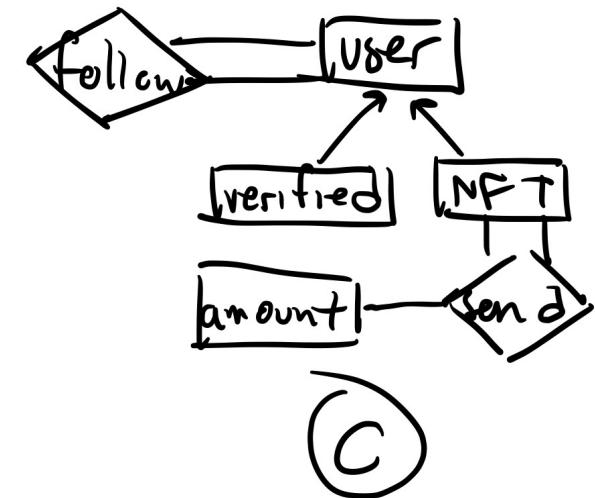
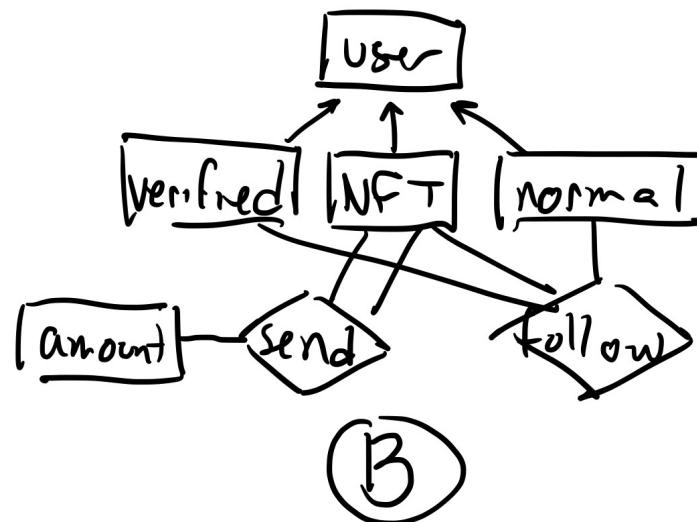
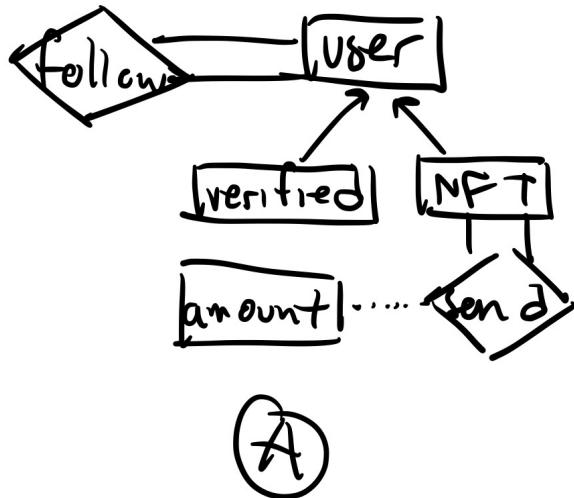
B



C

<http://w4lll.github.io/quiz.html>

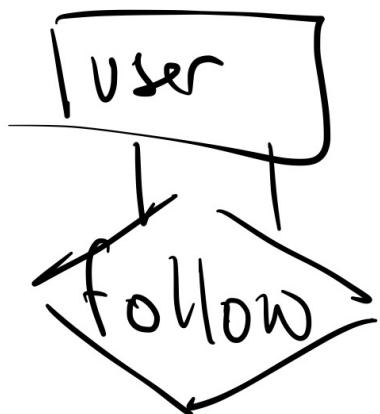
Any **user** can follow another **user**, but only **NFT** users can send **bitcoin** to each other.



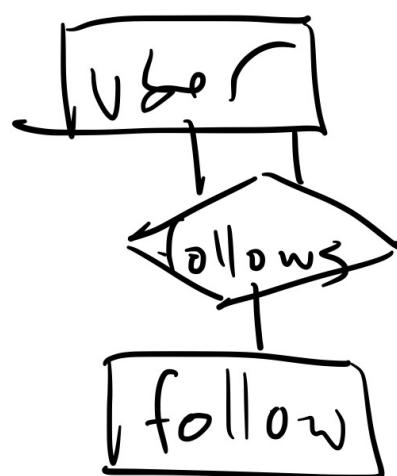
<http://w4lll.github.io/quiz.html>

Suppose Eugene follows BigDataMemes on Twitter.

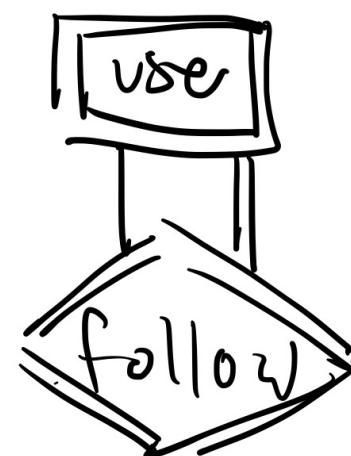
If Eugene deletes his account, BigDataMemes should not have Eugene as a follower.



(A)



(B)



(C)

The Relational Model

Eugene Wu

Background

Relational model: most widely used data model in the world

Legacy Models

IMS hierarchical/Nested

CODASYL network

Other common models:

Key-Val, Matrix, Data frame

Great video of the history: <https://www.youtube.com/watch?v=z8L202FlmD4>

Overview of Key Relational Principles

Data redundancy (or how to avoid it)

Physical data independence

Can change physical storage w/out changing programs

Logical data independence

Can change schema without changing programs

High level languages

Historical Context

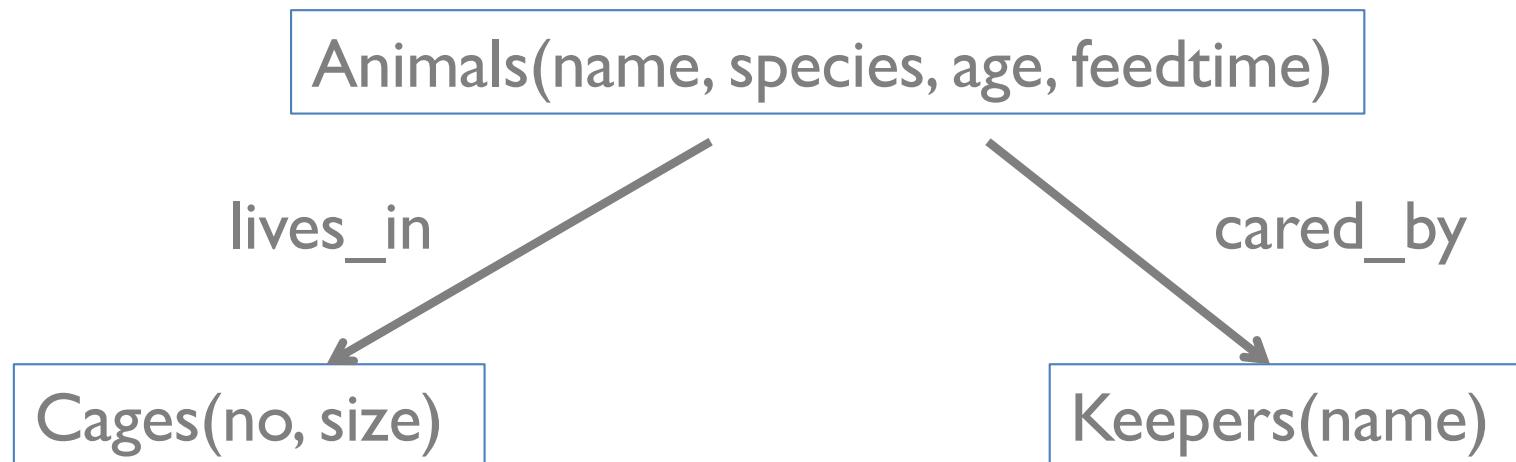
Hierarchical model (IMS)

Network model (CODASYL)

Relational model (SQL/QUEL)

70s

80-90s



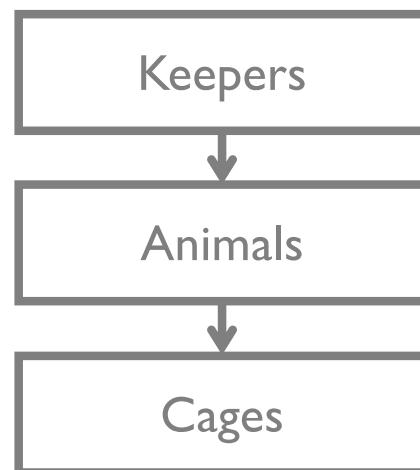
This notation is to give an example of an alternative syntax, NOT the ER diagram taught or used in this class

Hierarchical Model (IMS, circa 1968)

Segment types (objects)

Segment instances (records)

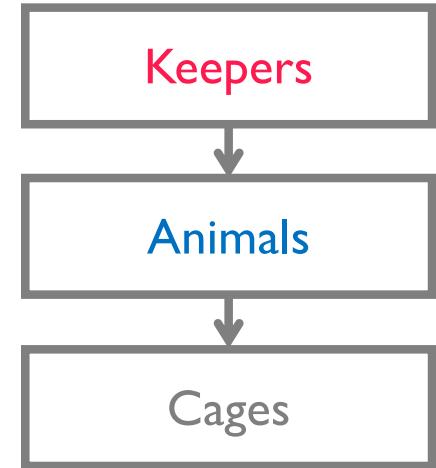
Segment types form a tree



Data Stored Depth First

Jane (HSK 1)
Bob, iguana, (2)
1, 100ft², (3)
Joe, student, (4)
1, 100ft², (5)

```
{ id: 1
  name: Jane
  animals: [
    { id: 2
      name: Bob
      species: iguana,
      cages: [
        {id: 3,
         no: 1,
         size: 100
       }]
    },
    ...
  ],
  ...
}
```



What's repeated?

Inconsistencies possible, lack of protections

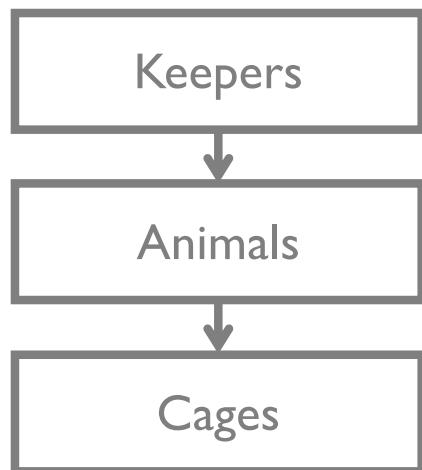
Similar to JSON

Can't Avoid Redundancy

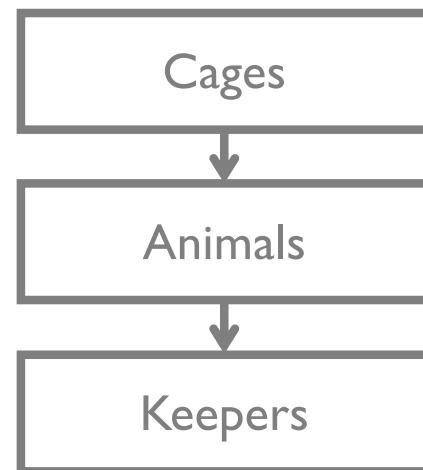
Segment types (objects)

Segment instances (records)

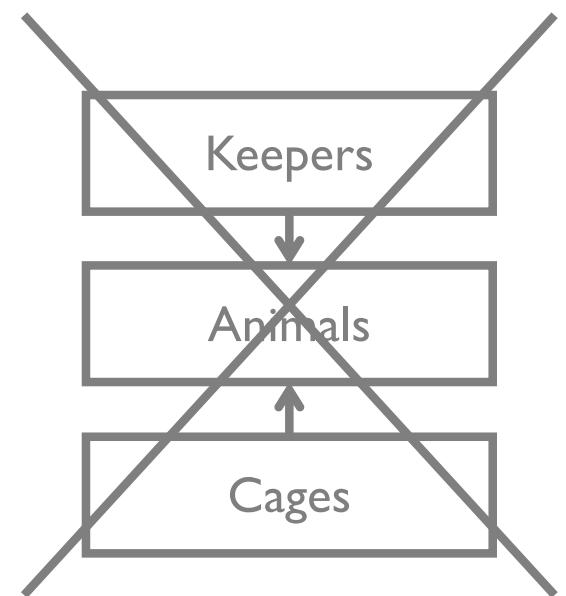
Segment types form a tree



Repeats cage data if
>1 animals in a cage



Repeats keeper data if
keeper cares for >1 animals



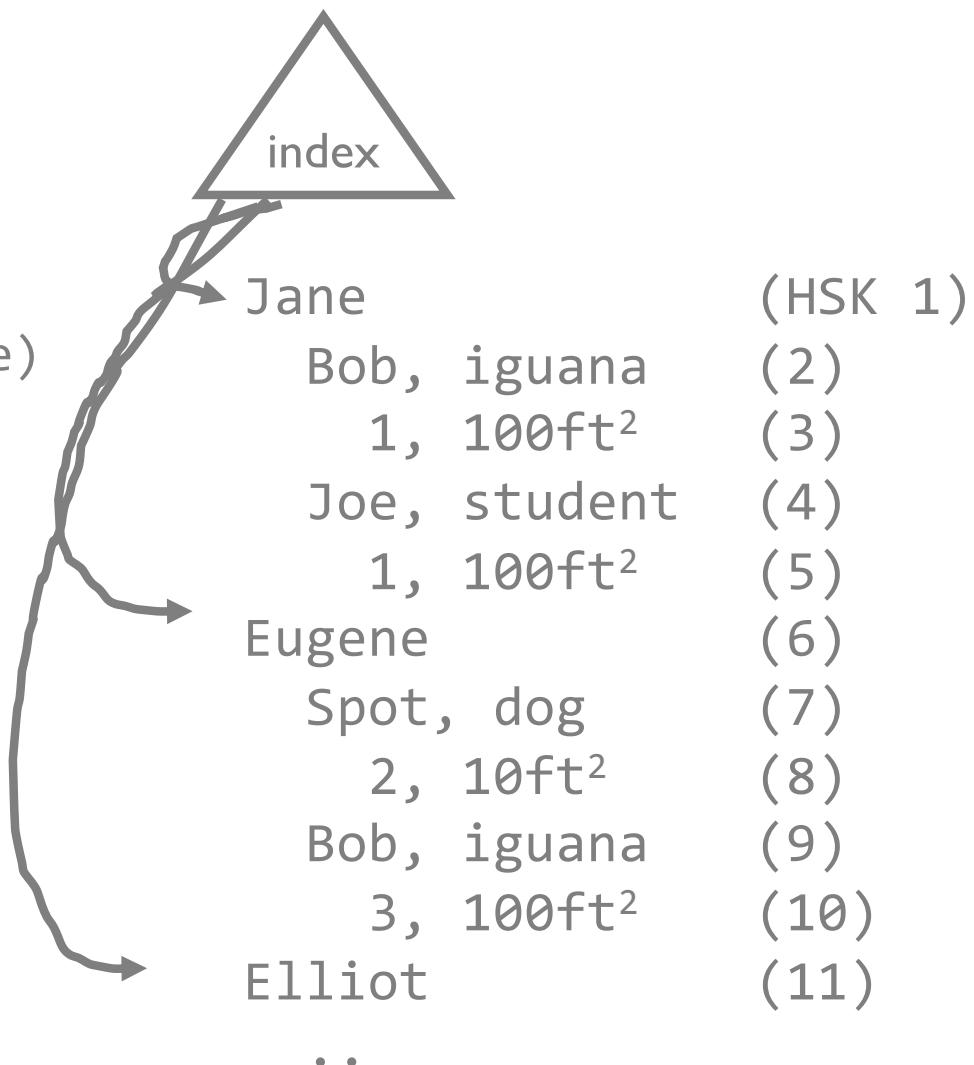
DAG != Tree

Hierarchical “Querying”

Fetch cages that Eugene manages

```
Goto(Keeper, name = Eugene)  
Until no more records  
    cage = NextInParent(Cage)  
    print cage.no
```

Output:



Hierarchical “Querying”

Fetch cages that Eugene manages

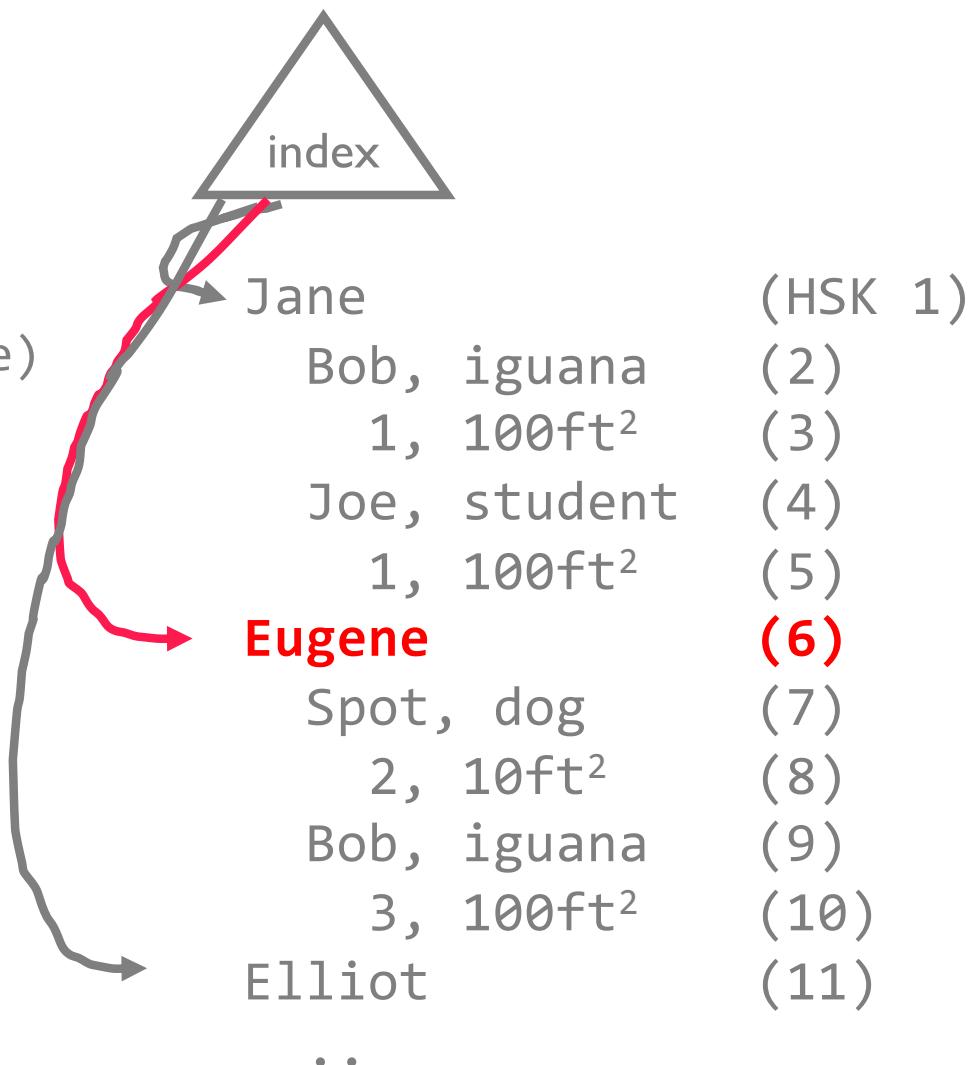
Goto(Keeper, name = Eugene)

Until no more records

 cage = NextInParent(Cage)

 print cage.no

Output:



Hierarchical “Querying”

Fetch cages that Eugene manages

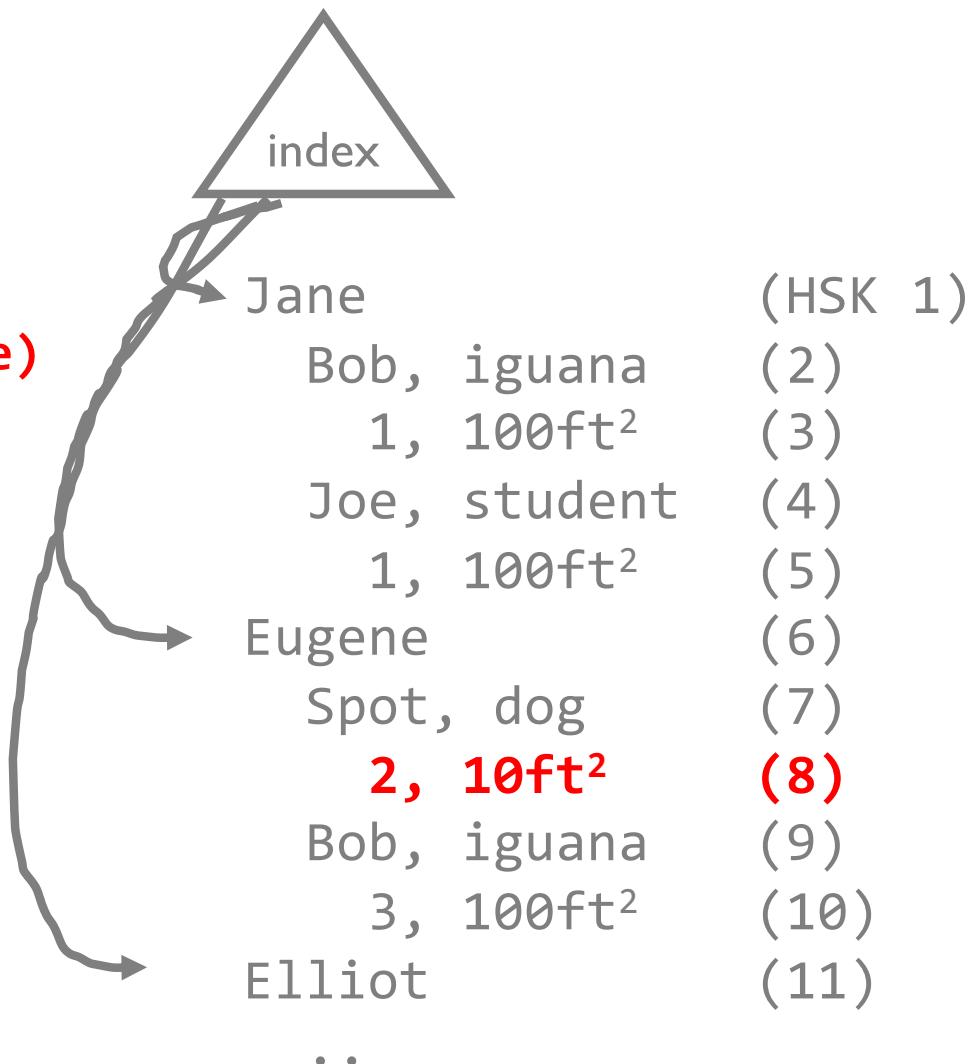
Goto(Keeper, name = Eugene)

Until no more records

cage = NextInParent(Cage)

print cage.no

Output:



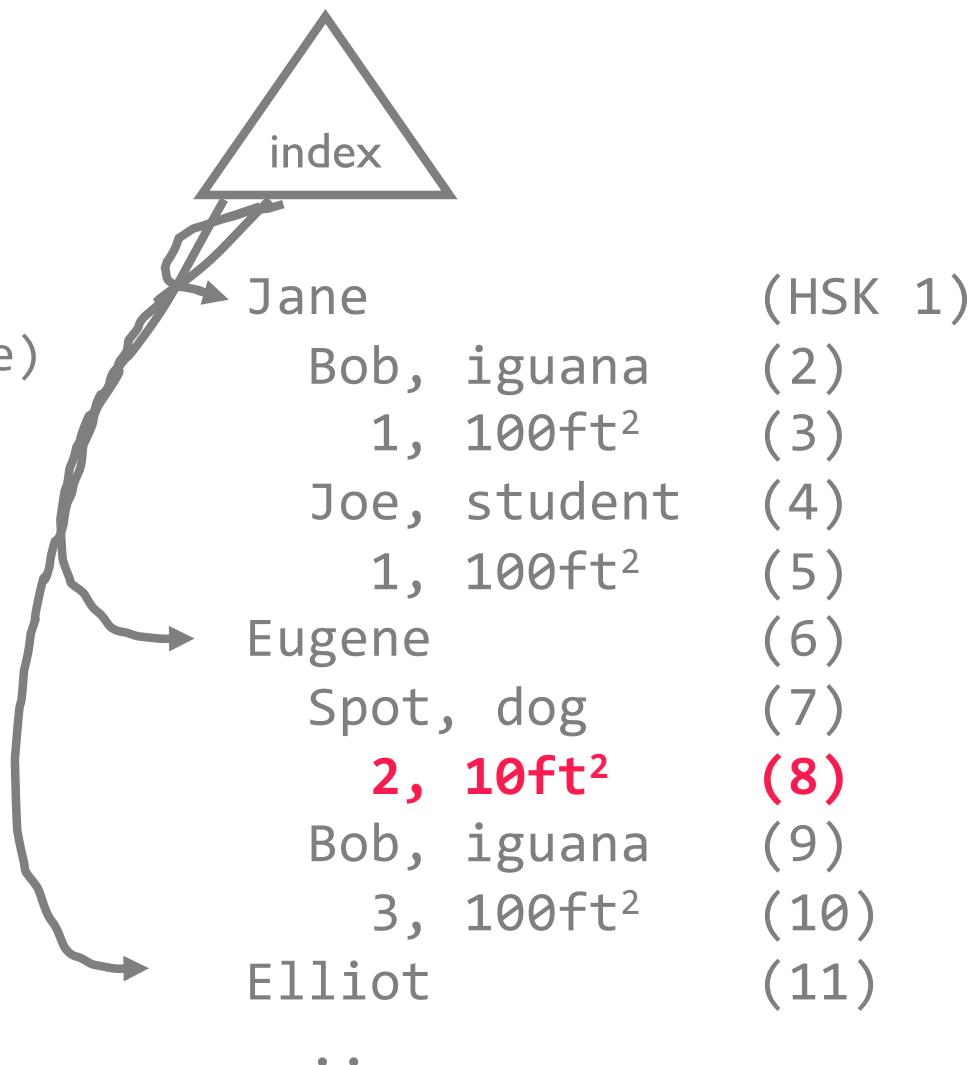
Hierarchical “Querying”

Fetch cages that Eugene manages

```
Goto(Keeper, name = Eugene)  
Until no more records  
    cage = NextInParent(Cage)  
print cage.no
```

Output:

2



Hierarchical “Querying”

Fetch cages that Eugene manages

Goto(Keeper, name = Eugene)

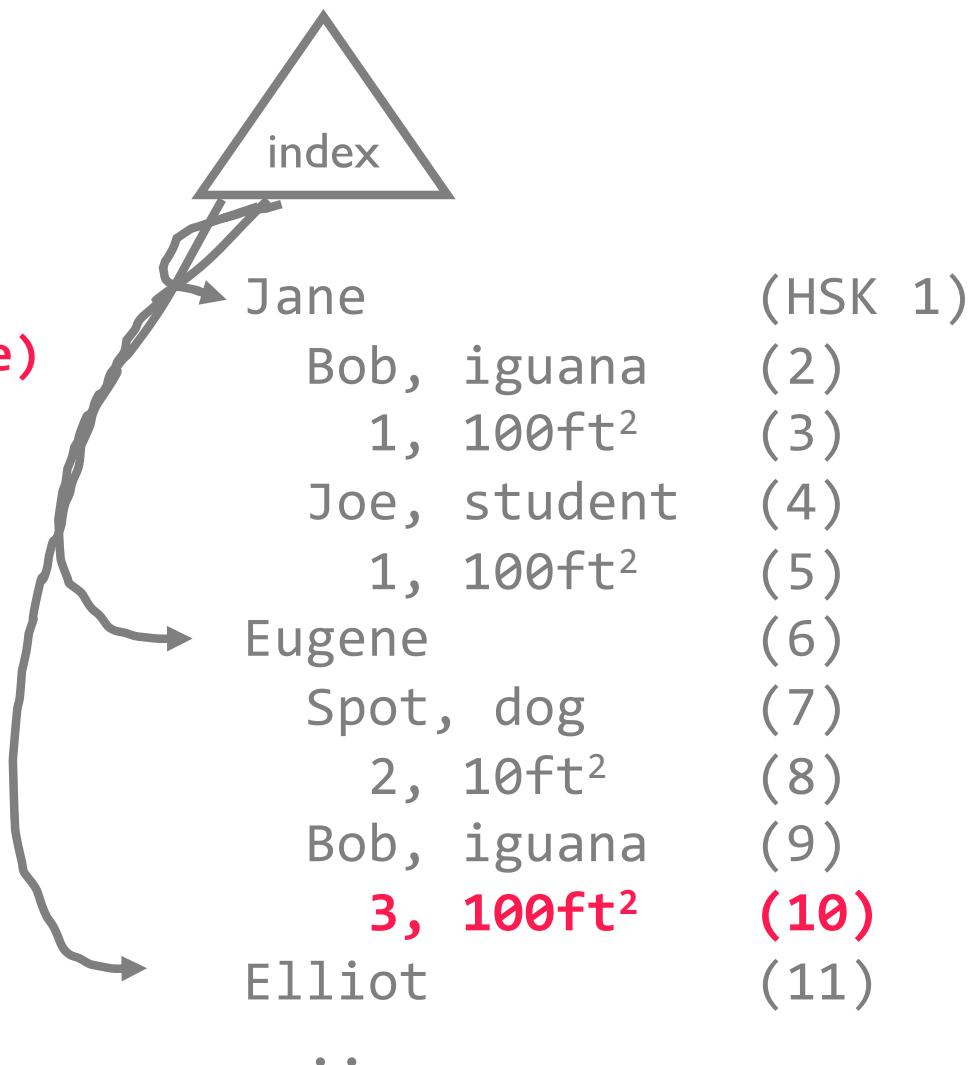
Until no more records

cage = NextInParent(Cage)

print cage.no

Output:

2



Hierarchical “Querying”

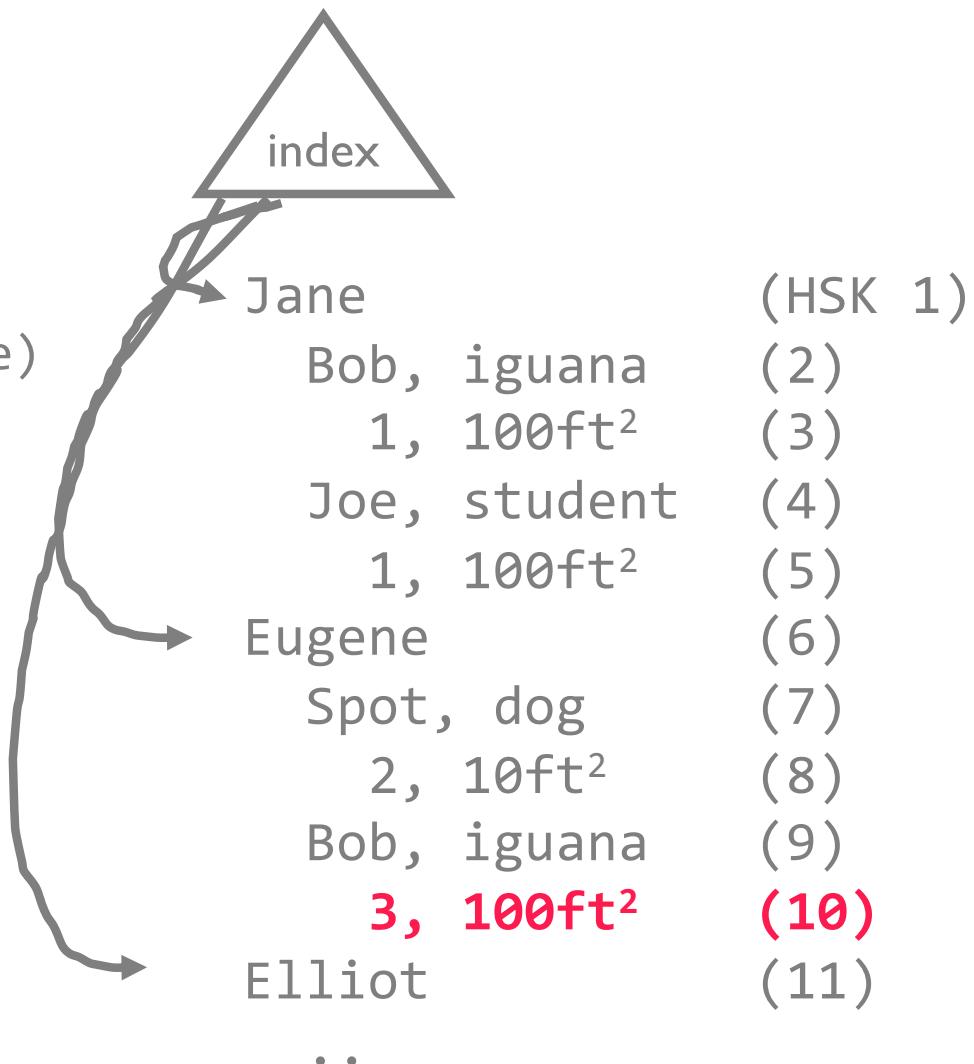
Fetch cages that Eugene manages

```
Goto(Keeper, name = Eugene)  
Until no more records  
    cage = NextInParent(Cage)  
print cage.no
```

Output:

2

3



Hierarchical “Querying”

Fetch cages that Eugene manages

Goto(Keeper, name = Eugene)

Until no more records

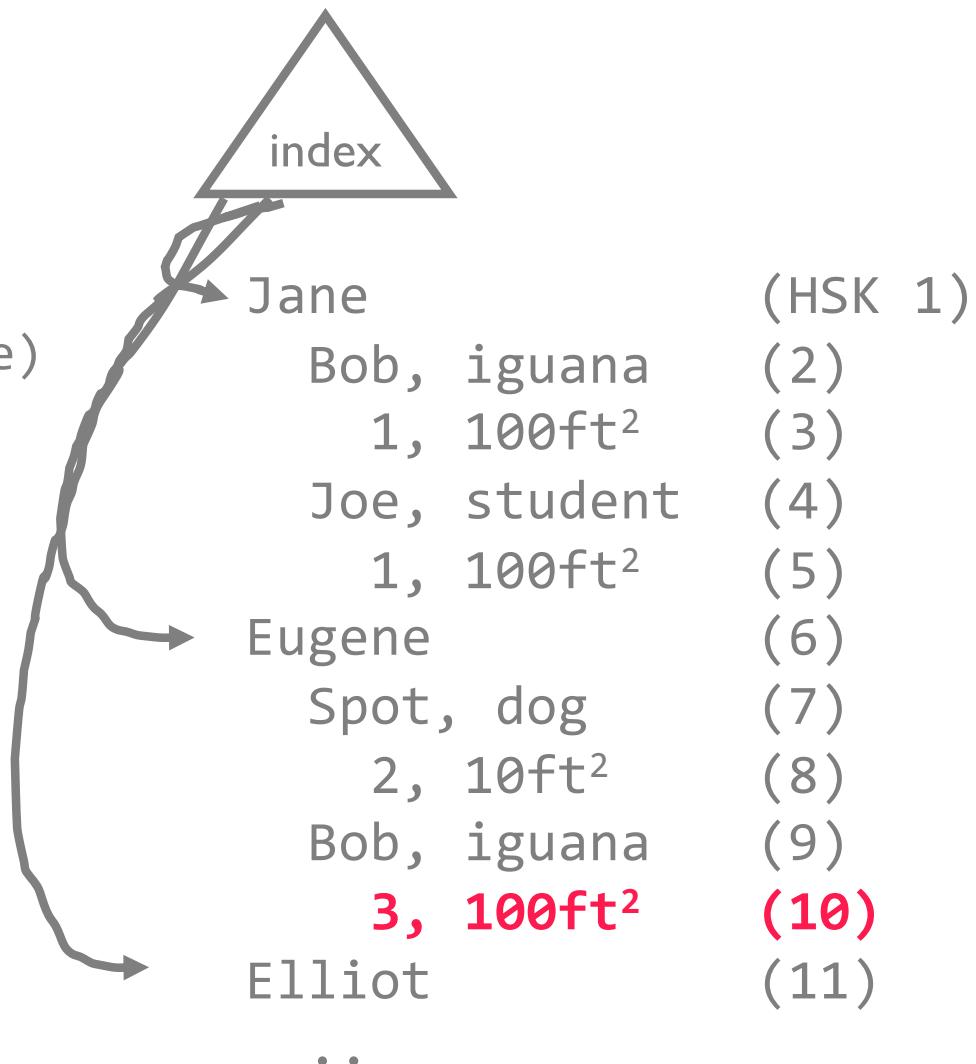
cage = NextInParent(Cage)

print cage.no

Output:

2

3



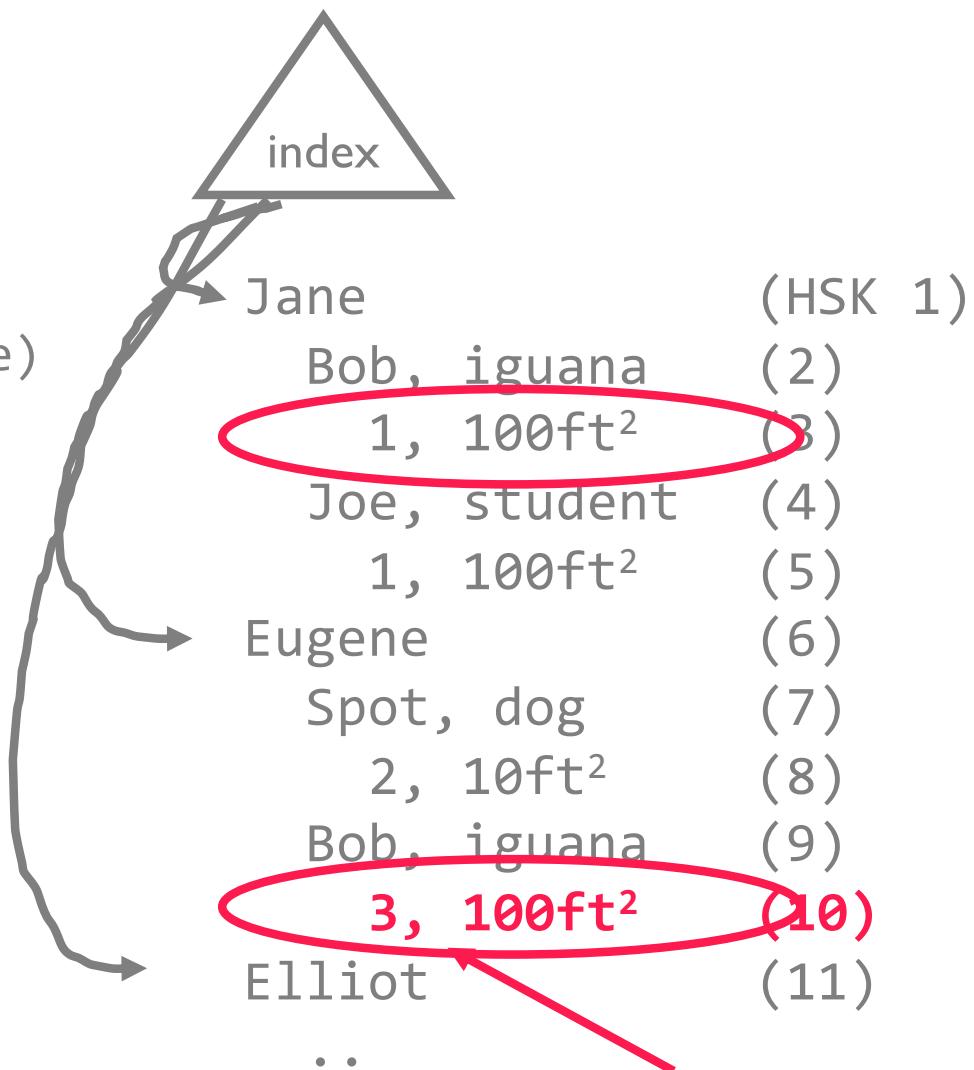
Hierarchical “Querying”

Fetch cages that Eugene manages

```
Goto(Keeper, name = Eugene)  
Until no more records  
    cage = NextInParent(Cage)  
    print cage.no
```

Output:

2
3



Notice the redundant and inconsistent data

Hierarchical “Querying”

Find Keepers of Cage 1

```
keeper = Goto(Keeper)
```

```
NextInParent(Cages, no=1)
```

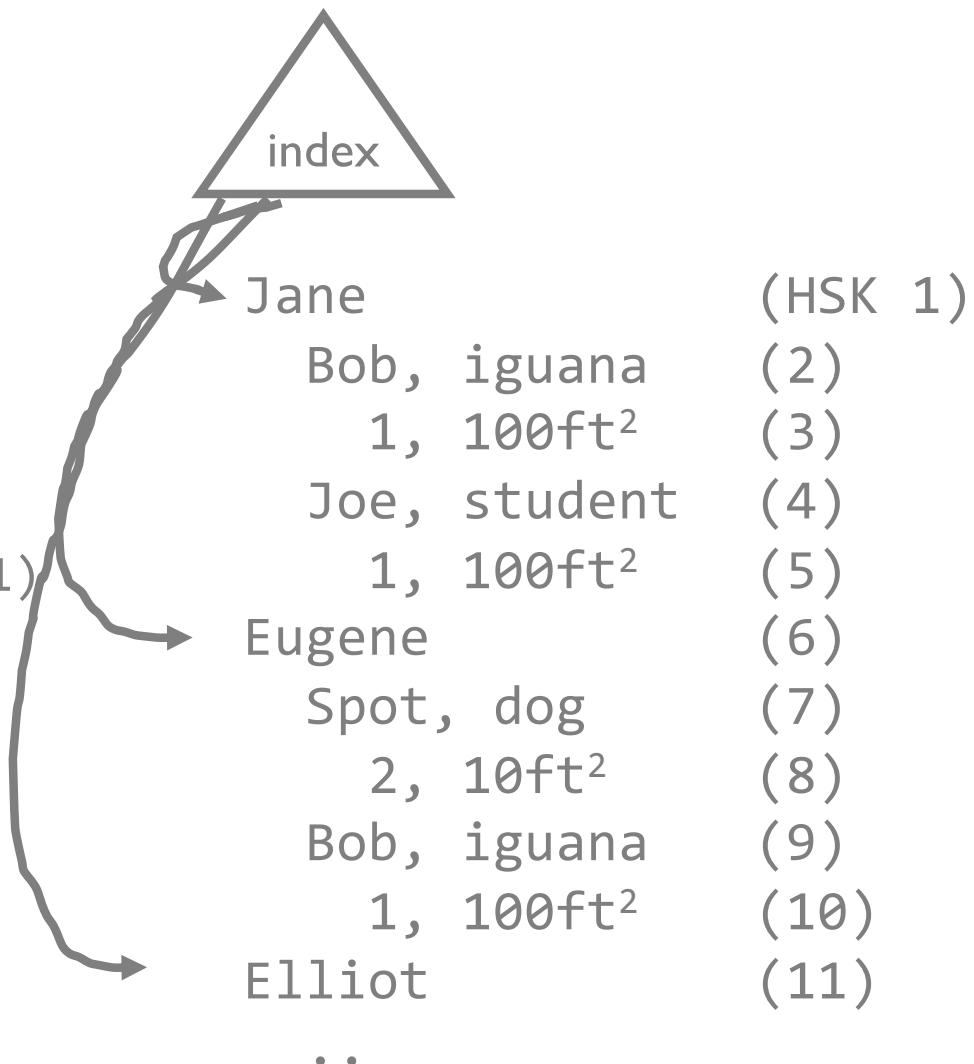
```
print keeper
```

```
Until no more records
```

```
    keeper = Next(Keeper)
```

```
    NextInParent(Cages, no=1)
```

```
    print keeper
```



Problems

Duplicates data

Low level programming interface

Almost no physical data independence

Change root from tree to hash index causes programs with GN
on root to *fail*

Inserts into sequential root structures disallowed

Lacks logical data independence

Changing schema requires changing program

Violates many desirable properties
of a proper DBMS

More Problems

Schema changes require program changes because
pointers after GetNext() calls now different

But...schemas change all the time

Keepers now responsible for a whole cage

Hummingbirds require multiple feedings

Merge with another zoo

Network Models (CODASYL, 1969)

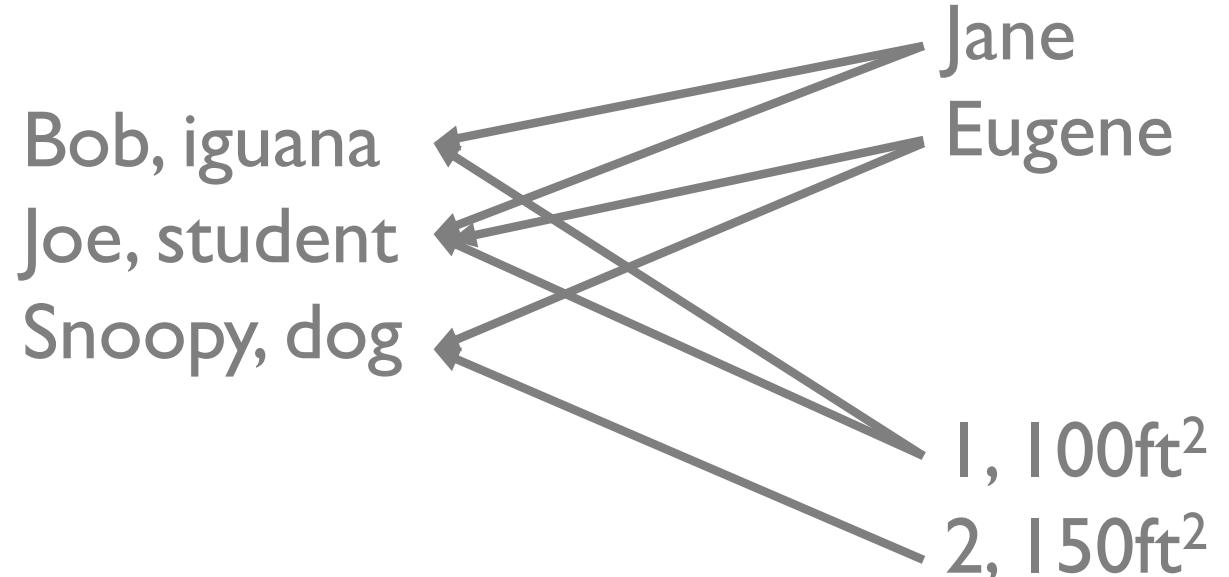
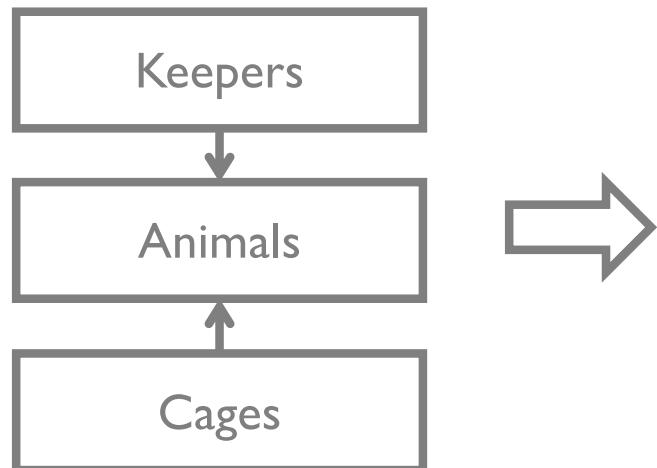
"Conference/Committee on Data Systems Languages"

Abstraction

Record Types connected by named sets (basically hash tables)

Connections are one way. Can't relate with same record type

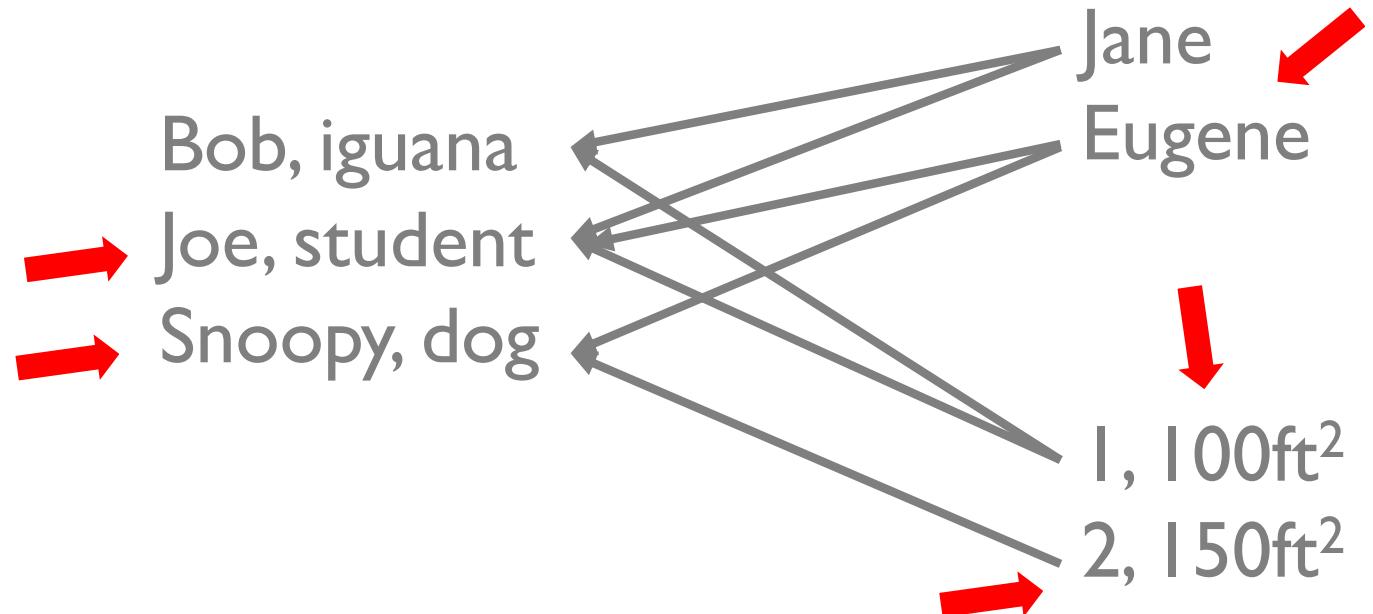
Modeled as a graph



Network Models (CODASYL, 1969)

```
Find Keeper (name = 'Eugene')  
until no more animals  
    Find next Animal in cares_for  
    until no more cages  
        Find next Animal' in cares_for  
        is Animal = Animal'?
```

...



Network Models: Problems

Very complex due to navigational programming

Strictly harder than hierarchical programming
hard to hire these programmers!

No data independence

Poor implementations e.g., main memory only

Bad trade off: increased programmer pain in order to model non-hierarchical data

Relational Model (1970)

Ted Codd, 1970

Reaction to CODASYL

Key properties

1. simple representation
2. set oriented model
3. no physical data model needed

Information Retrieval

A Relational Model of Data for
Large Shared Data Banks

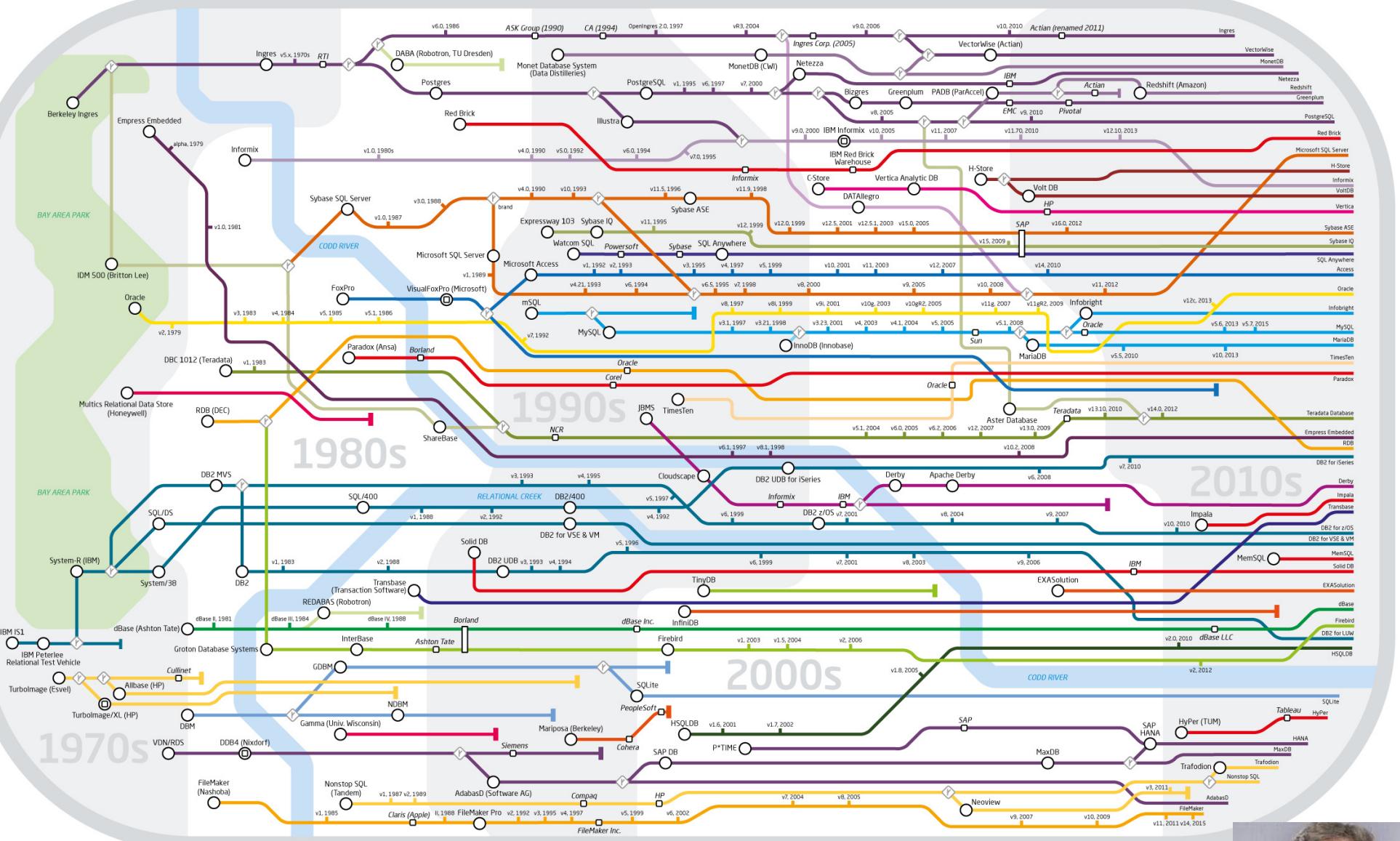
E. F. CODD

IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models

Genealogy of Relational Database Management Systems





Franck Pachot @FranckPachot · Sep 10

Ent

Replying to @MongoDB

Real friends will patiently explain to n00b friends why relational model was invented. #consistency #security #abstraction #encapsulation #normalization #joins #multi-purpose model.

seas.upenn.edu/~zives/03f/cis...

Future users of large data banks must be protected from

[Docs Home](#) / [MongoDB Atlas](#) / [Query Federated Data](#) / [SQL](#)

Query with Atlas SQL Statements

The page gives example Atlas SQL queries. You'll find basic examples that use SQL syntax to query collections, as well more advanced ones that use `FLATTEN` and `UNWIND` to work with nested data.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, is introduced.



The Birth of SQL & the Relational Database



Asianometry 712K subscribers

Subscribe

9K

...

Share

All

Computer programming

Presentations

...

K12 Private,

<https://www.youtube.com/watch?v=z8L202FlmD4>

Roadmap

History lesson

DDLs: Data definition language

Create schemas and constraints

DMLs: Data Manipulation Language Selection

Select, insert, delete, update data

ER → Relational Model

Theory: Relational Model

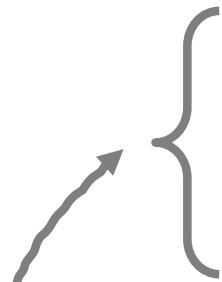
Relation defined by schema and contains set of tuples

Schema: relation name and **ordered list** of (attribute,type) pairs

an attribute/column



<u>sid</u>	name	login	age	gpa
1	eugene	ewu@cs	20	2.5
2	luis	luis@cs	20	3.5
3	ken	ken@cs	33	3.9



tuples: value for each attribute

Theory: Relational Model

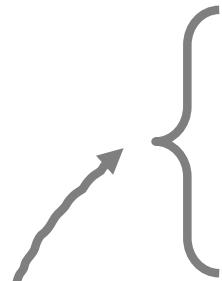
Relation defined by schema and contains set of tuples

`students(sid int, name text, login text, age int, gpa float)`

an attribute/column



<u>sid</u>	name	login	age	gpa
1	eugene	ewu@cs	20	2.5
2	luis	luis@cs	20	3.5
3	ken	ken@cs	33	3.9

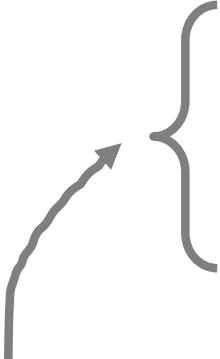


tuples: value for each attribute

Theory: Relational Model

Relation defined by schema and contains set of tuples

Schema is metadata



<u>sid</u>	name	login	age	gpa
1	eugene	ewu@cs	20	2.5
2	luis	luis@cs	20	3.5
3	ken	ken@cs	33	3.9

Tuples are data

Theory: Relational Model

Relation defined by schema and contains set of tuples

<u>sid</u>	name	login
1	eugene	ewu@cs
2	luis	luis@cs
3	ken	ken@cs

=?

<u>sid</u>	login	name
1	ewu@cs	eugene
2	luis@cs	luis
3	ken@cs	ken

Not Same

Theory: Relational Model

Relation defined by schema and contains set of tuples

“Relation instance”: schema and a **set** of specific tuples
unordered, no duplicates

Database set of relations

“Database instance”: set of relation instances

sid	name	login	age	gpa
1	eugene	ewu@cs	20	2.5
2	luis	luis@cs	20	3.5
3	ken	ken@cs	33	3.9

relation instance

Can the values of two columns be the same?

Can the values of two rows be the same?

Going Back to the Examples

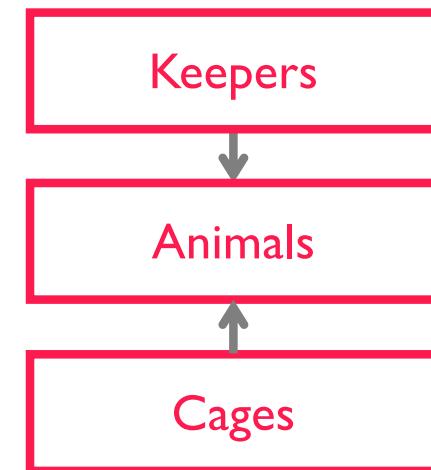
Keepers(id int, name text)

caresfor(keeper_id int, animal_id int)

Animals(id int, name text, species text)

livesin(animal_id int, cage_no)

Cages(no int, size int)



Terminology

Formal Name	Synonyms
Relation	Table
Tuple	Row, Record
Attribute	Column, Field
Domain	Type

Theory vs Practice

Theory: relational model is set-oriented

Practice: databases allow duplicates (multi-set semantics)

When discussing relational model, we will focus on the *theory*

The following slides describe integrity constraints using concrete syntax

DDL: CREATE TABLE

Create the Students Relation

Note: attribute domains are defined & enforced by DBMS

Create the Enrolled relation

```
CREATE TABLE Students(  
    sid int,  
    name text,  
    login text,  
    age int,  
    gpa float  
)
```

```
CREATE TABLE Enrolled(  
    sid int,  
    cid int,  
    grade char(2)  
)
```

Integrity Constraints (ICs)

def: a condition that is true for *any* instance of the database

Often specified when defining schema
DBMS enforces ICs at all times

An instance of a relation is **legal** if it satisfies all declared ICs
Programmer doesn't have to worry about data errors!
e.g., data entry errors

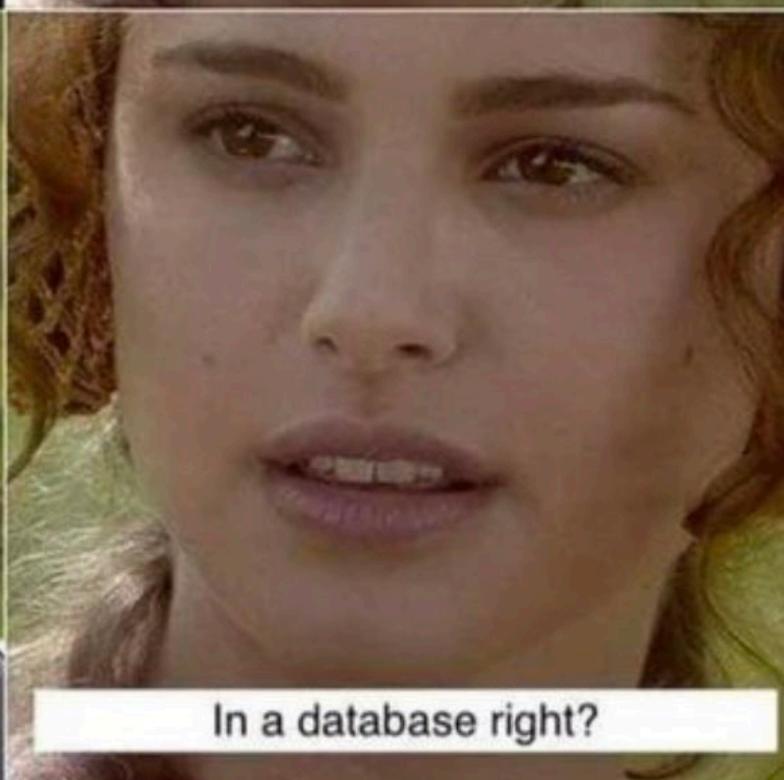
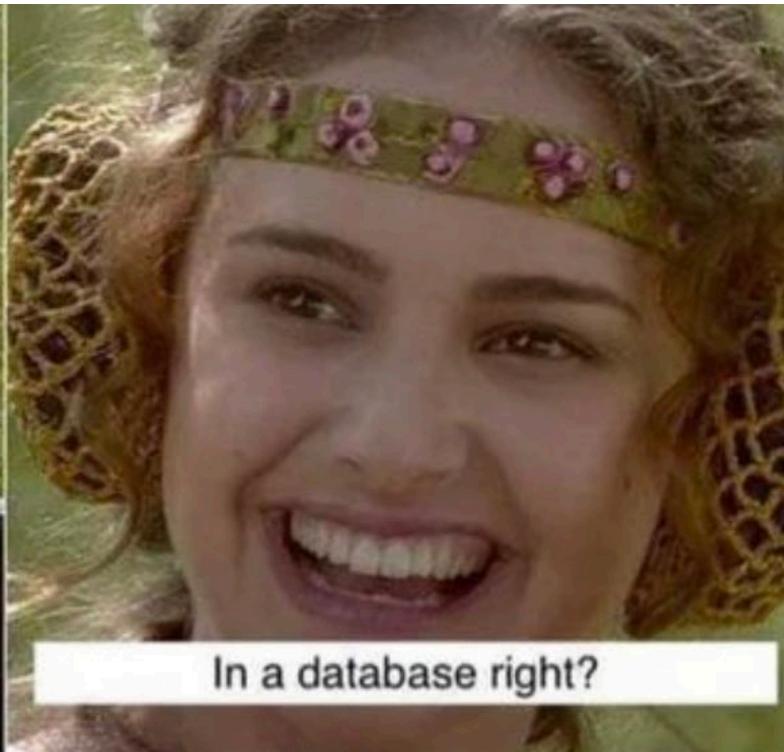
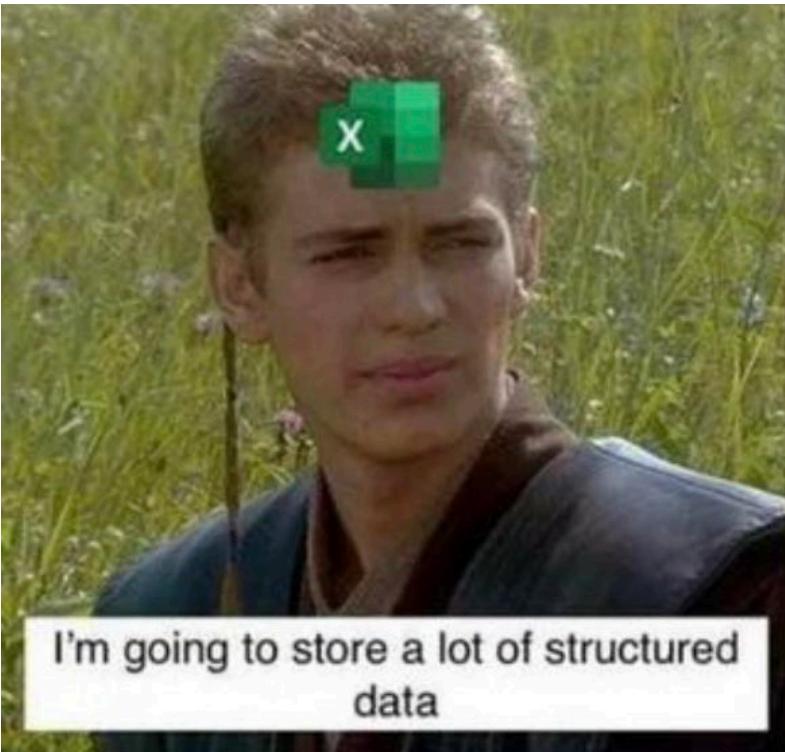
PostgreSQL documentation great resource
<https://www.postgresql.org/docs/current/ddl-constraints.html>

Domain Constraints (attr types)

```
CREATE TABLE Students(  
    sid int,  
    name text,  
    login text,  
    age int,  
    gpa float  
)
```

NULL Constraints

```
CREATE TABLE Students(  
    sid int NOT NULL,  
    name text,  
    login text,  
    age int,  
    gpa float  
)
```



POSTGRESQL



OF LEMONS
ANIZATION

MONGODB



Administrivia

HWI Part 2 releasing today

Project I Part I approval phase

Office Hours and Links

- [OH Calendar](#)
- Proj1 Part 1 Appointments
 - [Eugene Wu's appointment page](#)
 - [Jordyn Jaffe's appointment page](#)
 - [Vareesh's appointment page](#)
 - [Jerry's appointment page](#)
 - [Riz Chen's appointment page](#)
 - [Haonan's appointment page](#)

--

L3

L4

L5

L6

L7

-

Wu's Clues



Template: G have a relationship with E called eg

Courses have a relationship with Courses called prereq

Users have a relationship with Tickets called purchased

Courses are prereqs of other Courses

Users purchase Tickets

Candidate Keys

Set of fields is a *candidate key (or just Key)* for a relation if:

1. No two distinct tuples have same values in all key fields
2. This is untrue if you remove any attribute in the key
3. Key fields cannot be NULL

A set of attributes is a *superkey* if some subset of its attributes is a key. What's a trivial superkey?

If >1 candidate keys in relation, developer picks one as *primary key*
DBMS uses primary key as default when it needs to choose a key

sid is key for Students

is name a key?

what is (sid, gpa)?

Primary and Candidate Keys

UNIQUE & PRIMARY KEY key words

Be careful with ICs:

Each student can enroll in
a course only once

What does this say?

```
CREATE TABLE Enrolled(  
    sid: int,  
    cid: int,  
    grade: char(2),  
    PRIMARY KEY (sid, cid)  
)
```

```
CREATE TABLE Enrolled(  
    sid: int,  
    cid: int,  
    grade: char(2),  
    PRIMARY KEY (sid),  
    UNIQUE (cid, grade)  
)
```

Primary and Candidate Keys

UNIQUE + NOT NULL == field is a key

PRIMARY KEY implies UNIQUE NOT NULL

DBMS uses primary key as default for e.g., foreign key references

```
CREATE TABLE Students(  
    sid int NOT NULL UNIQUE,  
    name text,  
    login text,  
    age int,  
    gpa float,  
    PRIMARY KEY (sid)  
)
```

Primary and Candidate Keys

UNIQUE + NOT NULL == field is a key

PRIMARY KEY implies UNIQUE NOT NULL

DBMS uses primary key as default for e.g., foreign key references

```
CREATE TABLE Students(  
    sid int PRIMARY KEY,  
    name text,  
    login text,  
    age int,  
    gpa float  
)
```

Default Candidate Keys

sqlite, duckdb, oracle

rowid: “hidden” unique id column

postgreSQL

ctid: location of tuple. May change arbitrarily!

Foreign Keys

def: set of fields in Relation R_i used to refer to tuple in R_j via R_j 's keys (a logical pointer)

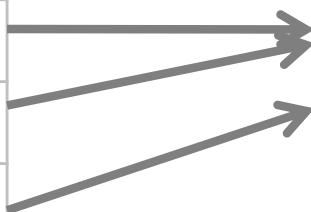
```
CREATE TABLE Enrolled(
    sid int,
    cid int,
    grade char(2),
    PRIMARY KEY (sid, cid),
    FOREIGN KEY (sid) REFERENCES Students(sid)
)
```

Enrolled

sid	cid	grade
1	2	A
1	3	B
2	2	A+

Students

sid	name
1	eugene
2	luis



Foreign Keys

def: set of fields in Relation R_i used to refer to tuple in R_j via R_j 's keys (a logical pointer)

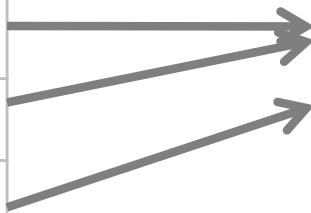
```
CREATE TABLE Enrolled(
    sid int REFERENCES Students(sid),
    cid int,
    grade char(2),
    PRIMARY KEY (sid, cid)
)
```

Enrolled

sid	cid	grade
1	2	A
1	3	B
2	2	A+

Students

sid	name
1	eugene
2	luis



Foreign Keys

def: set of fields in Relation R_i used to refer to tuple in R_j via R_j 's keys (a logical pointer)

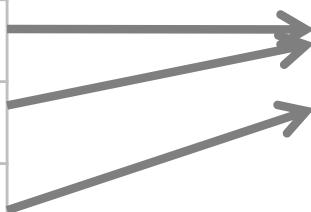
```
CREATE TABLE Enrolled(
    sid int REFERENCES Students,
    cid int,
    grade char(2),
    PRIMARY KEY (sid, cid)
)
```

Enrolled

sid	cid	grade
1	2	A
1	3	B
2	2	A+

Students

sid	name
1	eugene
2	luis



Foreign Keys

def: set of fields in Relation R_i used to refer to tuple in R_j via R_j 's keys (a logical pointer)

```
CREATE TABLE Donation(
    companyid int,
    classid int,
    amount float,
    PRIMARY KEY(companyid, classid)
)
CREATE TABLE Donation_Admin(
    daid int PRIMARY KEY,
    compid int,
    clsid int,
    FOREIGN KEY (compid, clsid) REFERENCES
        Donation(companyid, classid)
)
```

Referential Integrity

A database instance has *referential integrity* if all foreign key constraints are enforced
no “dangling references”

Examples where referential integrity is not enforced?

HTML links

Yellow page listing

Restaurant menus

CHECK Constraints

Boolean constraint expression added to schema

Very powerful mechanism.

Evaluated for each tuple, rejects tuple if false

```
CREATE TABLE Enrolled(  
    sid int,  
    cid int,  
    grade char(2),  
    CHECK (  
        grade IN ('A', 'B', 'C', 'D', 'F')  
    )  
)
```

How to Enforce Integrity Constraints

Run checks anytime database changes

On INSERT

What if new Enrolled tuple refers to non-existent student?

Reject insertion

On DELETE (many options)

What if Students tuple is deleted?

Delete dependent Enrolled tuples

Reject deletion

Set Enrolled.sid to default value or **null**

(null means ‘unknown’ or ‘inapplicable’ in SQL)

Where do ICs come from?

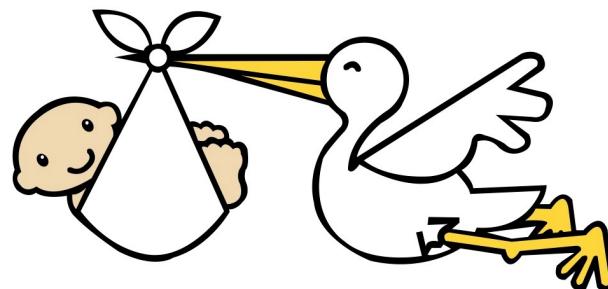
Based on application semantics and use cases

Can check if database instance satisfies ICs

IC is a statement over *all possible instances*

Can't infer ICs by staring at an instance

Key and foreign key ICs are most common, more general table and database constraints possible as well.



(mostly) More Powerful than ER Constraints

Functional dependencies

A dept can't order two distinct parts from the same supplier.

Can't express this wrt ternary Contracts relationship.

Used to further refine ER models (called normalization)

Inclusion dependencies

Domain of attrs in a table are subset of domain of different set of attrs (in same or different table)

Special case: ER model can express Foreign keys

At least 1 person must report to each manager.

General constraints

Each donation is less than 10% of the combined donations to all humanities courses. (but can be expensive to enforce)

What Can ER Express?

Key constraints,
participation constraints,
overlap/covering constraints

Some foreign key constraints as part of relationship set
Some constraints require general CHECK stmts
ER cannot express e.g., function dependencies at all
Constraints help determine best database design

Roadmap

History lesson

DDLs: Data definition language

Create schemas and constraints

DMLs: Data Manipulation Language

Select, insert, delete, update data

ER → Relational Model

DML: Introduction to Queries

Key strength of relational model
declarative querying of data

Queries are high level, readable
DBMS makes it fast, user don't need to worry

Precise semantics for relational queries
Lets DBMS choose different ways to run query while
ensuring answer is the same

INSERT/DELETE

Add a tuple

```
INSERT INTO Students(sid, name, login, age, gpa)  
VALUES (4, 'wu', 'wu@cs', 20, 5)
```

Delete tuples satisfying a predicate (condition)

```
DELETE FROM Students S  
WHERE S.name = 'wu'
```



Boolean predicate evaluated over
every tuple

Bulk Imports

varies by database system, read their docs

Postgresql

```
COPY into Students  
FROM '/users/ewu/students.csv' WITH CSV
```

SQLite

```
.mode csv  
.import /users/ewu/students.csv Students
```

DuckDB

```
CREATE TABLE Students AS  
SELECT * FROM read_csv_auto('/users/ewu/students.csv')
```

Basic SELECT

Get all attributes of
students younger than 21

```
SELECT *
FROM Students S
WHERE S.age < 21
```

sid	name	login	age	gpa
1	eugene	ewu@cs	20	2.5
2	luis	luis@cs	20	3.5
3	ken	ken@math	33	3.9

Basic SELECT

Get all attributes of
students younger than 21

```
SELECT *  
FROM Students S  
WHERE S.age < 21
```

Get only names

```
SELECT S.name  
FROM Students S  
WHERE S.age < 21
```

sid	name	login	age	gpa
1	eugene	ewu@cs	20	2.5
2	luis	luis@cs	20	3.5
3	ken	ken@math	33	3.9

Basic SELECT

Get all attributes of
students younger than 21

```
SELECT *  
FROM Students S  
WHERE S.age < 21
```

Get only names

```
SELECT S.name  
FROM Students S  
WHERE S.age < 21
```

sid	name	login	age	gpa
1	eugene	ewu@cs	20	2.5
2	luis	luis@cs	20	3.5
3	ken	ken@math	33	3.9

Single Table Semantics

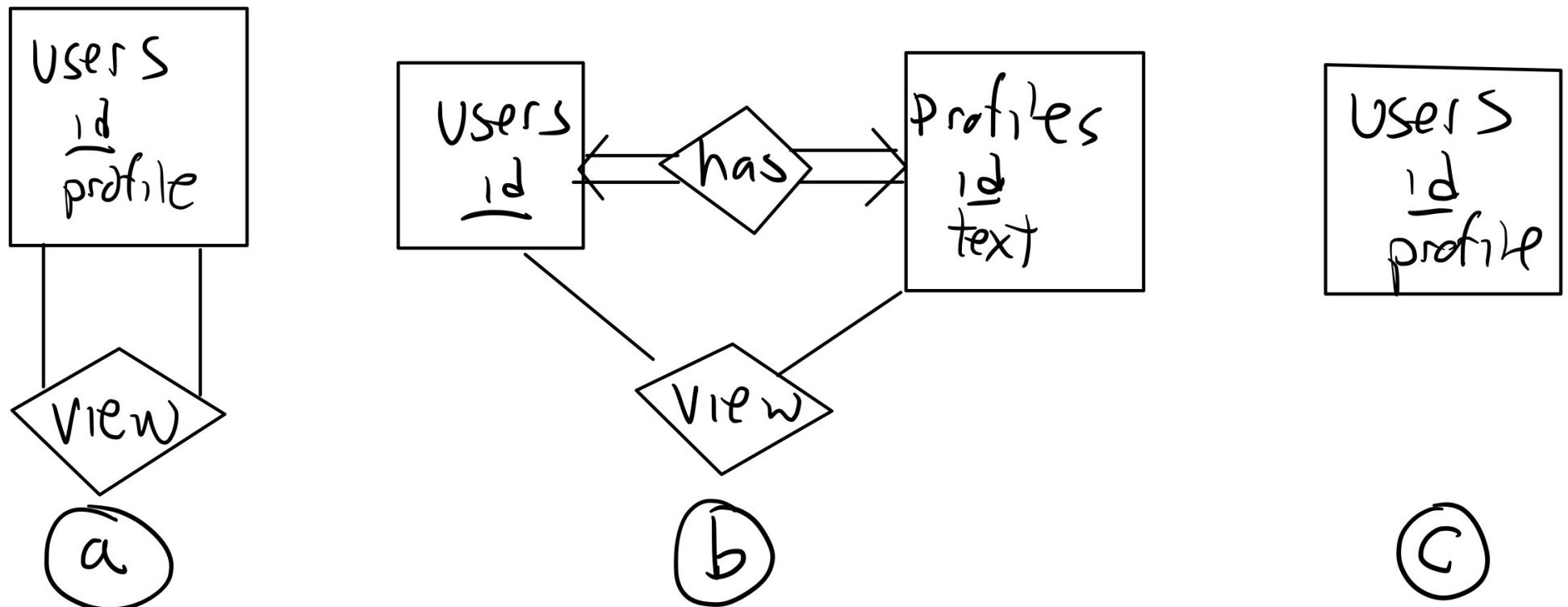
A *conceptual evaluation method* for previous query:

```
for row in Students           // FROM clause
  if predicate(row):          // WHERE clause
    emit desired fields      // SELECT clause
```

Actual evaluation will be *much* more efficient, but must produce the same answers.

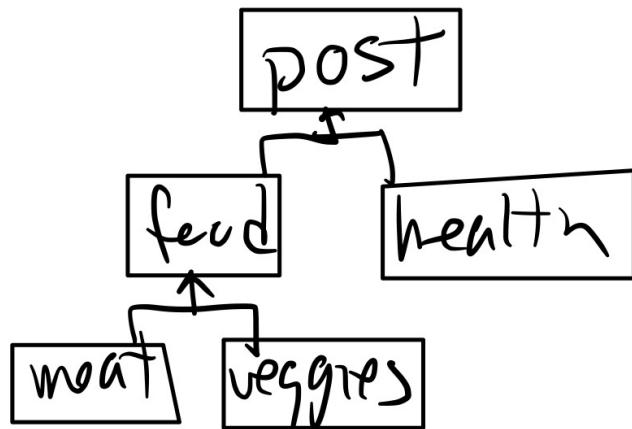
<http://w4lll.github.io/quiz>

The application lets **users view the profiles** of other **users**

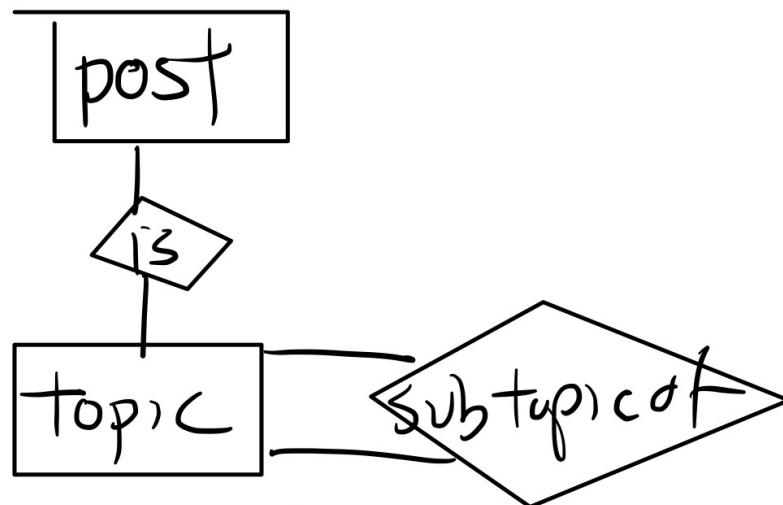


<http://w4lll.github.io/quiz>

A **post** can be about **food or health** and under **food** it can be about **meat or veggies**



(a)



(b)



(c)

<http://w4lll.github.io/quiz>

A user can **enroll** in at most one course.

What should be the constraints?

```
CREATE TABLE enrolls(  
    userid int REFERENCES users(id),  
    courseid int REFERENCES courses(id),  
    term int,  
    PRIMARY KEY (userid, courseid)  
)
```

Nothing needed

(A)

UNIQUE (userid)
NOT NULL userid

(B)

UNIQUE(userid)
NOT NULL userid
NOT NULL courseid

(C)

Translating ER → Relational Models

Approach:

Does ER version *allow and reject*
everything that the
Relational Model's version *allows and rejects?*

Entity Set → Relation

Course
cid
name
loc

```
CREATE TABLE Course(  
    cid int,  
    name text,  
    loc text,  
    PRIMARY KEY (cid)  
)
```

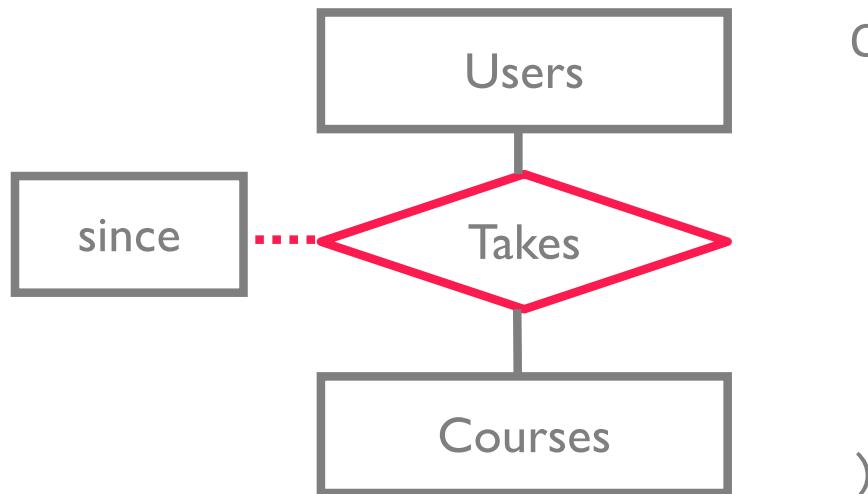
Relationship Set w/out constraint → Relation

Relation must include

Keys for each entity set as foreign keys

these form superkey for relation

All attributes of the relationship set



```
CREATE TABLE Takes(  
    uid int,  
    cid int,  
    since date,  
    PRIMARY KEY (uid, cid),  
    FOREIGN KEY (uid) REFERENCES Users,  
    FOREIGN KEY (cid) REFERENCES Courses  
)
```

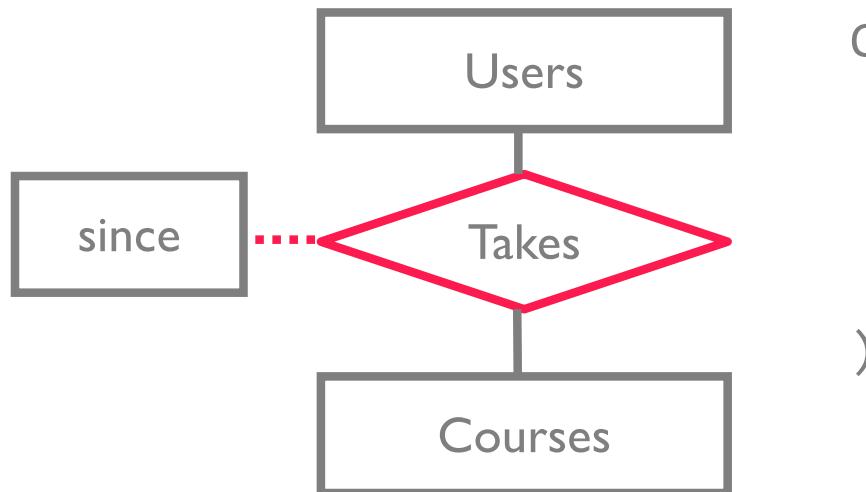
Relationship Set w/out constraint → Relation

Relation must include

Keys for each entity set as foreign keys

these form superkey for relation

All attributes of the relationship set



```
CREATE TABLE Takes(  
    uid int REFERENCES Users(uid),  
    cid int REFERENCES Courses(cid),  
    since date,  
    PRIMARY KEY (uid, cid)  
)
```

Short hand

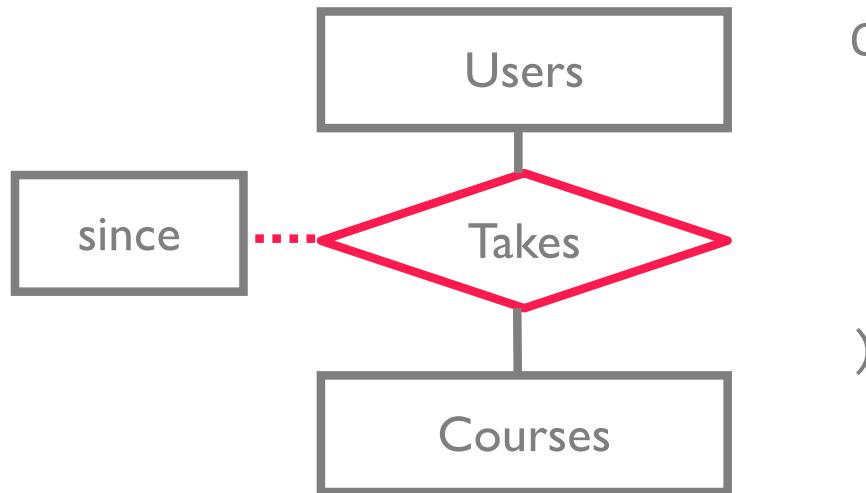
Relationship Set w/out constraint → Relation

Relation must include

Keys for each entity set as foreign keys

these form superkey for relation

All attributes of the relationship set



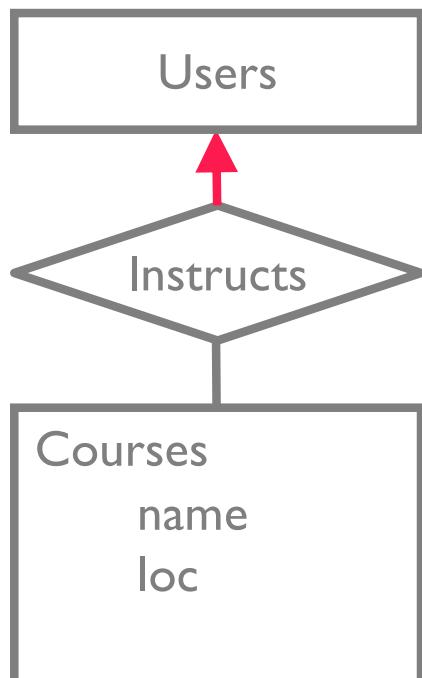
```
CREATE TABLE Takes(  
    uid int REFERENCES Users,  
    cid int REFERENCES Courses  
    since date,  
    PRIMARY KEY (uid, cid)
```

Shorter hand if
attribute names the same

Key Constraint → Relation

User and Courses are separate relations

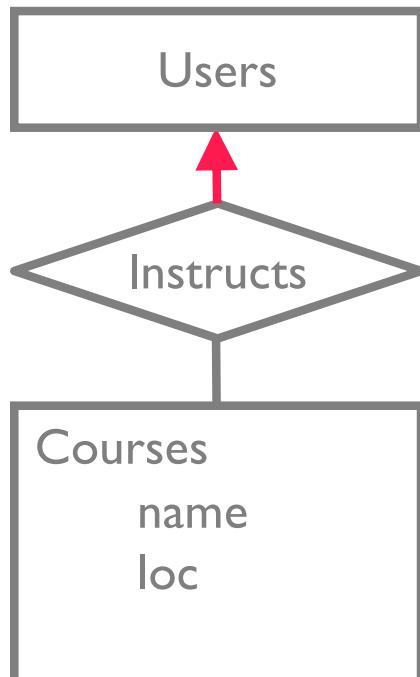
Note only cid is a Key... why not UNIQUE(uid, cid)?



```
CREATE TABLE Instructs(  
    cid int NOT NULL REFERENCES Courses,  
    uid int NOT NULL REFERENCES Users,  
    UNIQUE (cid, uid),  
  
    PRIMARY KEY cid  
)
```

Key Constraint → Relation

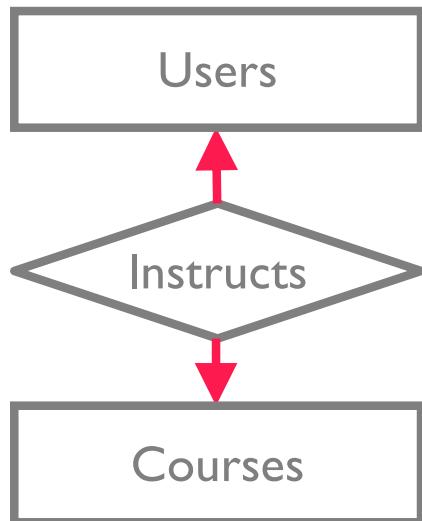
Alternatively combine Courses and Instructs
(this is the preferred way)



```
CREATE TABLE Course_Instructs(  
    cid int PRIMARY KEY,  
    name text,      // attr from Courses  
    loc text,       // attr from Courses  
    uid int REFERENCES Users  
)
```

Key Constraint → Relation

How to translate this ER diagram?



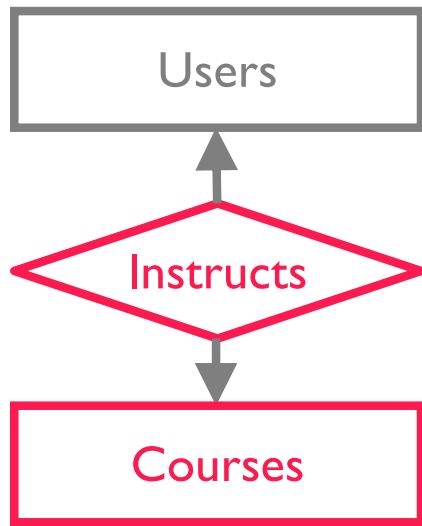
```
CREATE TABLE Course_Instructs(  
    ????  
)
```

Key Constraint → Relation

(wrong!)

Translation I

Merge Courses and Instructs



```
CREATE TABLE Course_Instructs(  
    cid int PRIMARY KEY  
    ... courses attributes ...
```

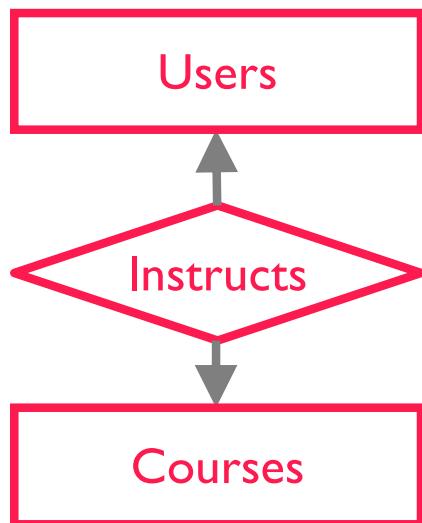
```
    uid int PRIMARY KEY REFERENCES Users  
)  
// course cannot have 0 users  
// cannot have 2 primary keys
```

Key Constraint → Relation

(wrong!)

Translation 2

Merge Courses and Instructs



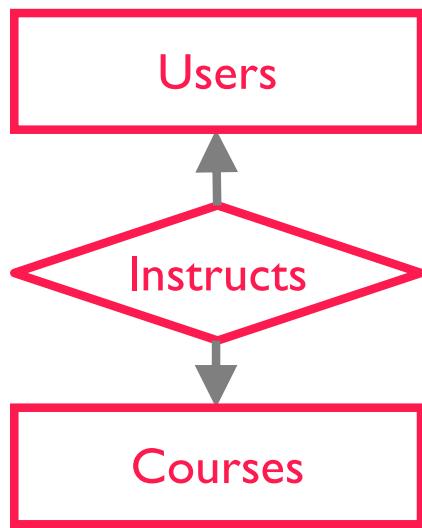
```
CREATE TABLE Course_Instructs(  
    cid int PRIMARY KEY,  
    ... courses attributes ...  
  
    uid int UNIQUE NOT NULL REFERENCES Users  
)  
// course cannot have 0 users
```

Key Constraint → Relation

(correct)

Translation 3

Merge Courses and Instructs



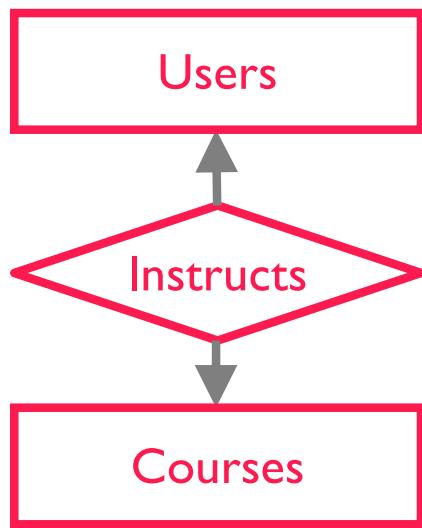
```
CREATE TABLE Course_Instructs(
    cid int PRIMARY KEY,
    ... courses attributes ...
    uid int UNIQUE REFERENCES Users
)
// course can have 0 users or 1 users
// user can have 0 or 1 course
```

Key Constraint → Relation

Translation 4

Merge all three relations.

Correct, but mixing many attrs can get messy



```
CREATE TABLE Course_Instructs_Users(  
    cid int UNIQUE,  
    uid int UNIQUE,  
    ... courses attributes ...  
    ... users attributes ...  
  
    CHECK (cid NOT NULL OR uid NOT NULL)  
)
```

Administrivia

Project I Part I

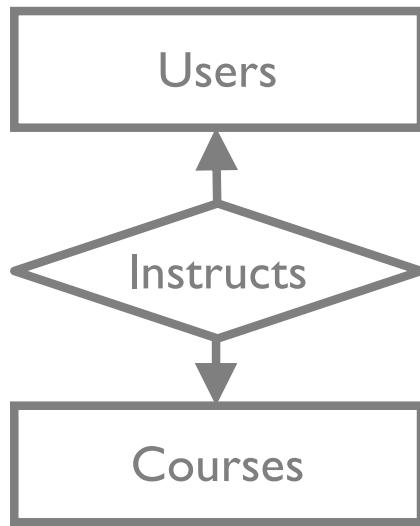
- submit “SQL Schema” means write the CREATE TABLE statements for your ER diagram
- We have extended Project I Part I to Friday 9/27 11:59pm EST
- Form will be up later today

Key Constraint → Relation

(correct)

Translation 5

Keep all three relations.



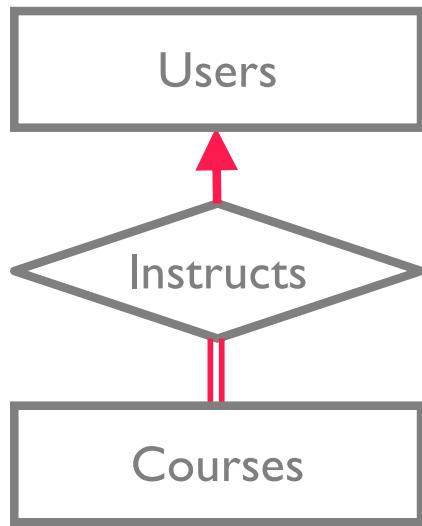
```
CREATE TABLE courses(cid, ...)  
CREATE TABLE users(uid, ...)
```

```
CREATE TABLE instructs(  
    uid int UNIQUE NOT NULL REFERENCES users,  
    cid int UNIQUE NOT NULL REFERENCES courses,  
    PRIMARY KEY (uid, cid)  
)
```

IF there is a relationship, it must satisfy constraints
in instructs table. But doesn't need to exist

Participation Constraint → Relation

Only exactly-one relationships can be correctly translated because no way to “force” a relationship tuple to exist.

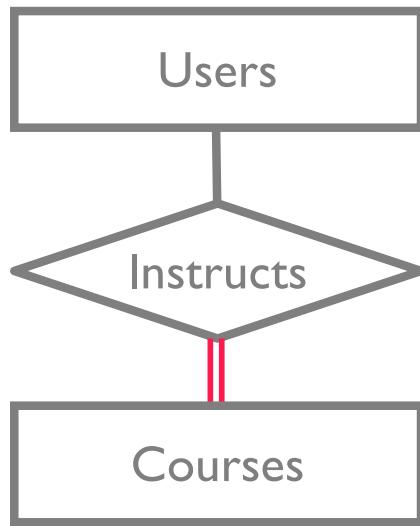


```
CREATE TABLE Course_Instructs(  
    cid int PRIMARY KEY,  
    uid int NOT NULL UNIQUE REFERENCES User  
    ... courses attributes ...  
  
    ON DELETE NO ACTION  
)
```

Participation Constraint → Relation

At-least-one can't be cleanly translated.

We only consider participation constraint with 1 entity set in a binary relationship (others need CHECK constraint)

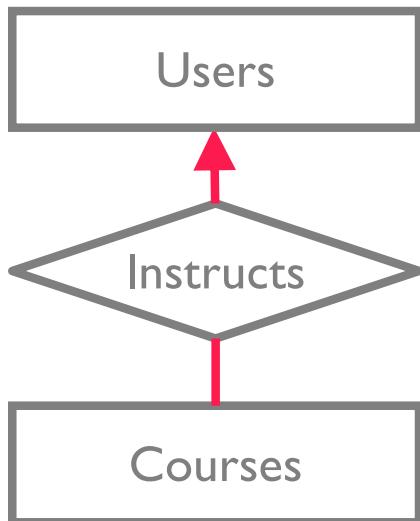


```
CREATE TABLE Course_Instructs(  
    cid int PRIMARY KEY,  
    uid int NOT NULL REFERENCES Users,  
    ... courses attributes ...  
)  
ON DELETE NO ACTION
```

*Not a fully correct
translation. Why?*

Participation Constraint → Relation

Using 3 tables can only enforce “0 or 1” relationships



```
CREATE TABLE Course()
```

```
CREATE TABLE Users()
```

```
CREATE TABLE Instructs(
```

```
    cid int PRIMARY KEY REFERENCES Course,  
    uid int NOT NULL REFERENCES Users
```

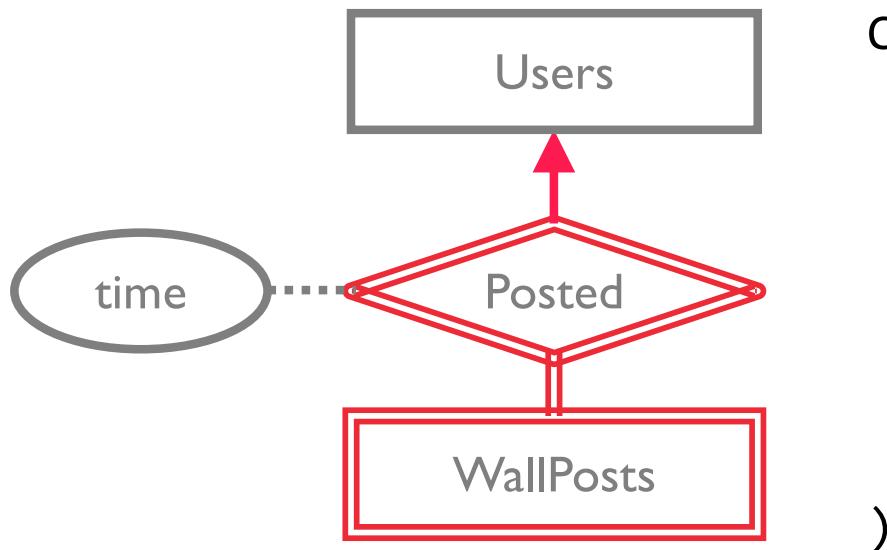
```
)
```

Weak Entity → Relation

Weak entity set and identifying relationship set are translated into a single table.

When owner entity is deleted, all owned weak entities also deleted.

post_title in this example is a partial key: it alone does not identify a given wall post! For example, if each user's first wall post is given post_title 1, then post_title does not uniquely identify a wall post since another user's first post may also be post_title = 1



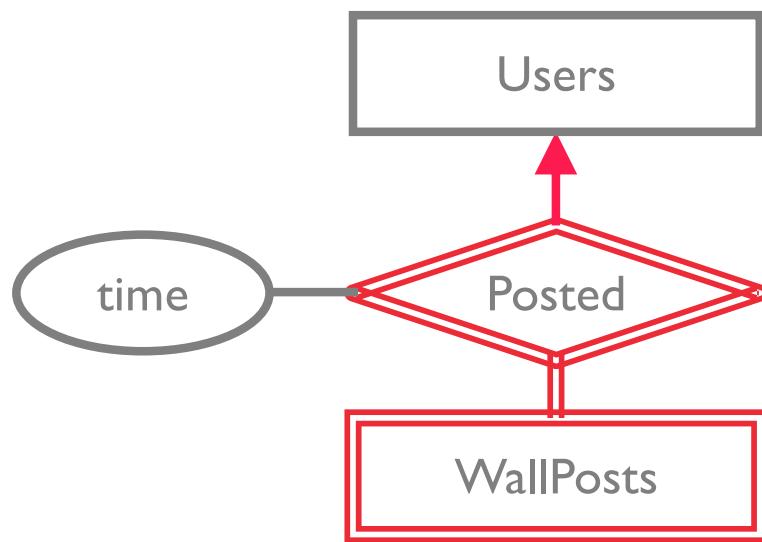
```
CREATE TABLE WallPosts_Posted(
    post_title text, // partial key
    post_text text,
    time DATE,
    uid int,
    PRIMARY KEY (post_title, uid),
    FOREIGN KEY (uid) REFERENCES Users
    ON DELETE CASCADE
)
```

Weak Entity → Relation

Weak entity set and identifying relationship set are translated into a single table.

When owner entity is deleted, all owned weak entities also deleted.

In practice, easy to make an auto incrementing id that is unique, so the primary key could simply be pid. In PostgreSQL, specified as serial. In this case we still need the ON DELETE CASCADE.



```
CREATE TABLE Wall_Posted(
    pid int serial,
    post_title text,
    post_text text,
    time DATE,
    uid int NOT NULL,
    PRIMARY KEY (pid),
    FOREIGN KEY (uid) REFERENCES Users
    ON DELETE CASCADE
)
```

A red arrow points from the word 'pid' in the 'PRIMARY KEY' clause to the 'pid' column in the CREATE TABLE statement. Another red arrow points from the word 'ON DELETE CASCADE' to the 'ON DELETE CASCADE' clause at the end of the CREATE TABLE statement.

Specialization Hierarchies

Option 1: Keep parent relation

B and C's attributes stored in A

Option 2: Keep parent and child relations

A stores only attributes of A

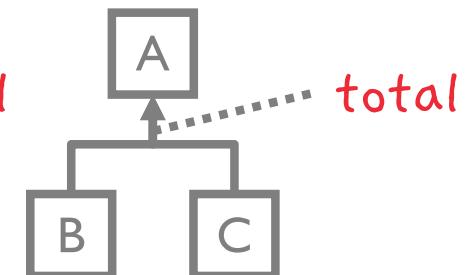
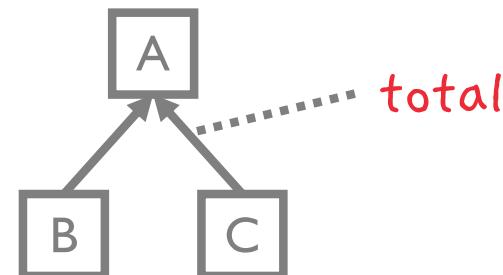
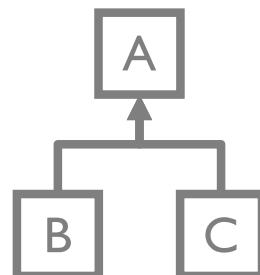
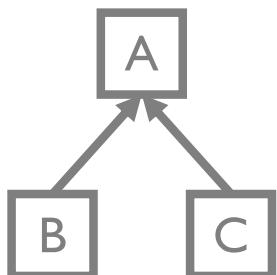
B and C store their own attributes

JOIN between child and base relations for all attributes

Option 3: Only keep child relations

B and C have copies of A's attributes

} Only under total specialization!



Specialization Hierarchies (Notes)

Each option has its own trade-offs in terms of

- what constraints it can express
- how complex the resulting schema is

The following examples focus on the main translation designs (number of relations).

Other constraints such as keys, not null, etc should still be considered

Analyze the consistency of a translation by

- adding records in the relations and seeing if they are allowed by the ER diagram,
- studying records that the relations disallow and if they are disallowed in the ER diagram
- and vice versa

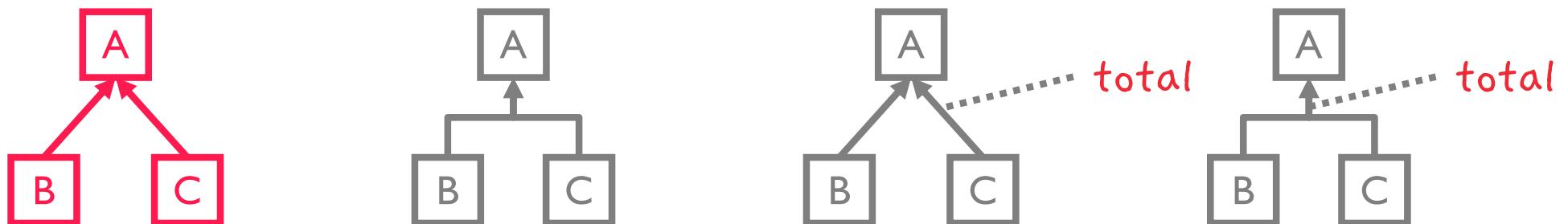
Specialization Hierarchies: Option I

A(a int PRIMARY KEY, name text)

B(b int PRIMARY KEY, a int NOT NULL REFERENCES A, ...)

C(c int PRIMARY KEY, a int NOT NULL REFERENCES A, ...)

This is incorrect. Two B records can be the same A



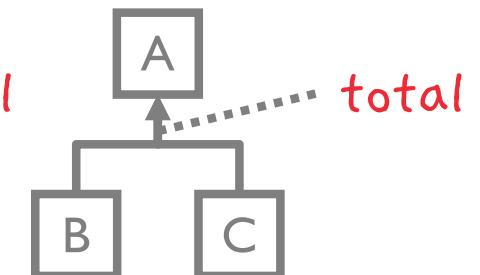
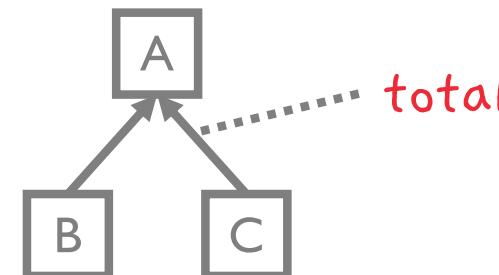
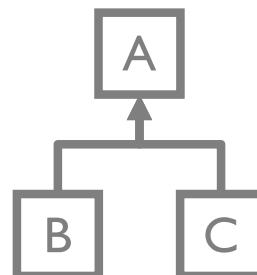
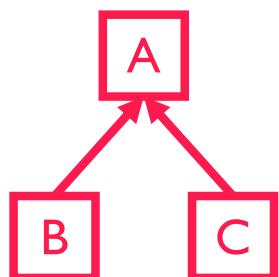
Specialization Hierarchies: Option I

A(a int PRIMARY KEY, name text)

B(b int PRIMARY KEY, a int NOT NULL UNIQUE REFERENCES A, ...)

C(c int PRIMARY KEY, a int NOT NULL UNIQUE REFERENCES A, ...)

Supports overlap = true, total = no



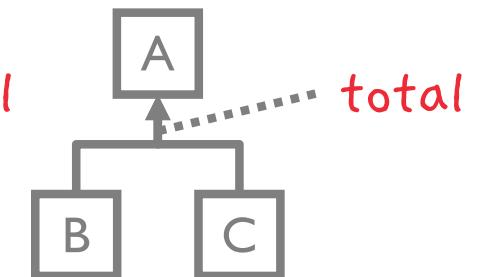
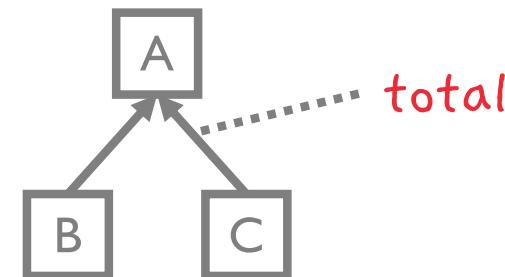
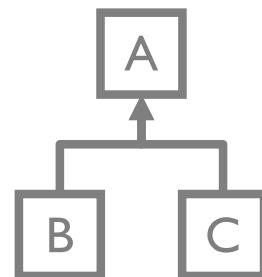
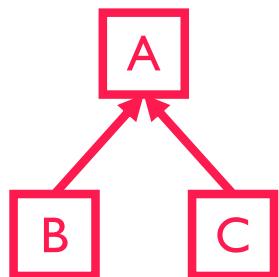
Specialization Hierarchies: Option I

A(a int PRIMARY KEY, name text)

B(a int PRIMARY KEY REFERENCES A, ..b attrs..)

C(a int PRIMARY KEY REFERENCES A, ..c attrs..)

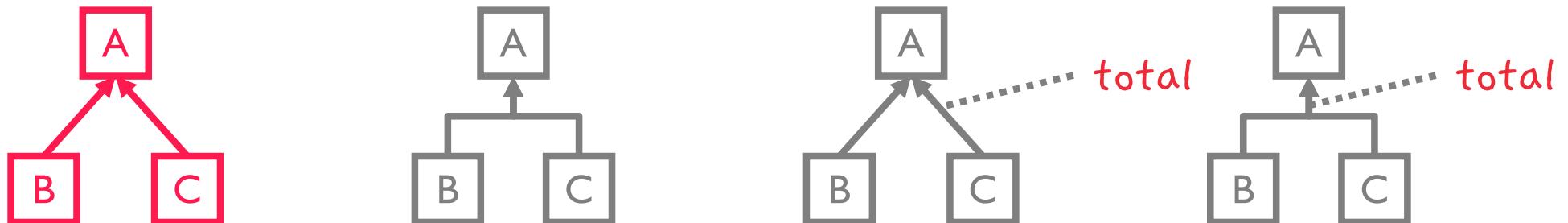
Supports overlap = true, total = no



Specialization Hierarchies: Option I

```
A(  
    a int PRIMARY KEY,  
    name text,  
    b int,  
    c int  
)
```

Supports overlap = true, total = no

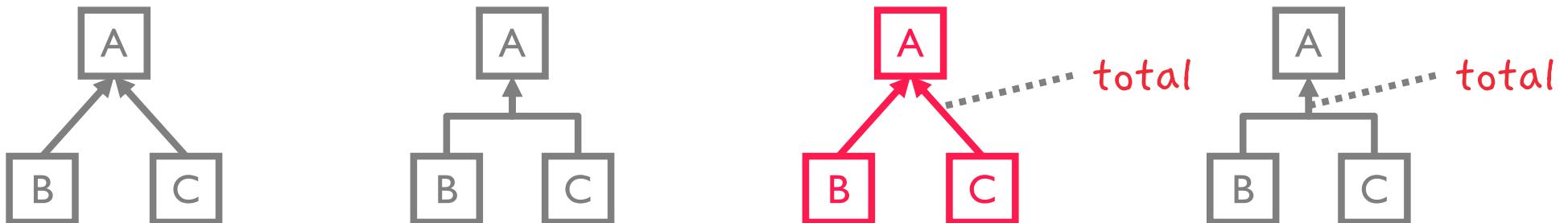


Specialization Hierarchies: Option 2

B(id int PRIMARY KEY, name text)

C(id int PRIMARY KEY, name text)

Enforces total specialization.
Can't have relationship with As in general



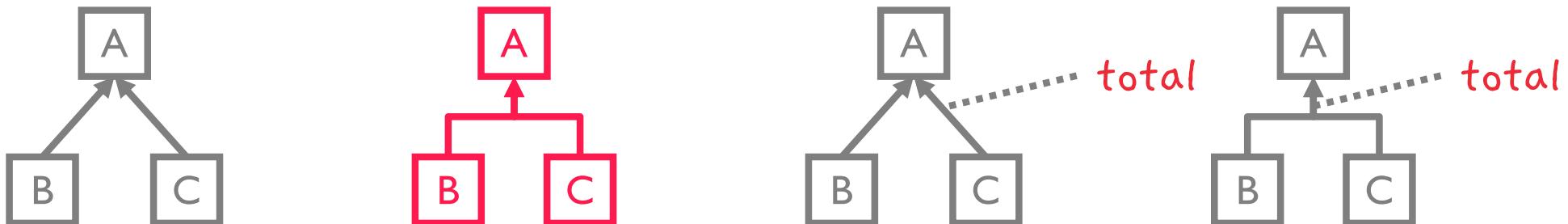
Specialization Hierarchies: Option 3

A(

```
a int PRIMARY KEY,  
name text,  
b int,  
c int,  
CHECK (not (b is not null and c is not null))
```

)

Supports overlap = no, total = no



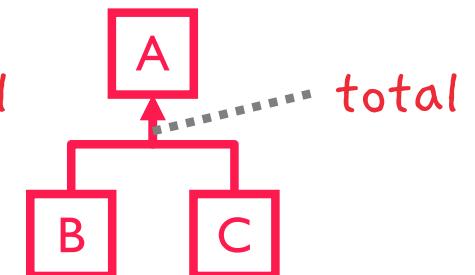
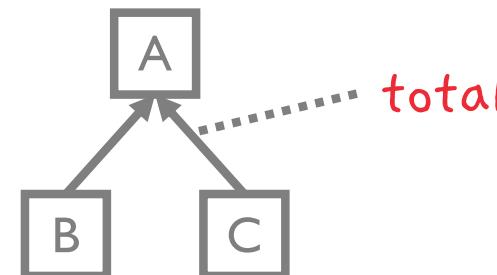
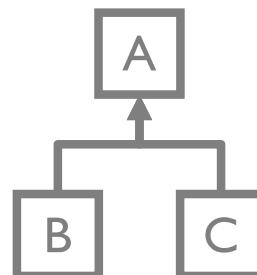
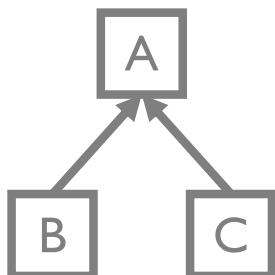
Specialization Hierarchies: Option 4

A(

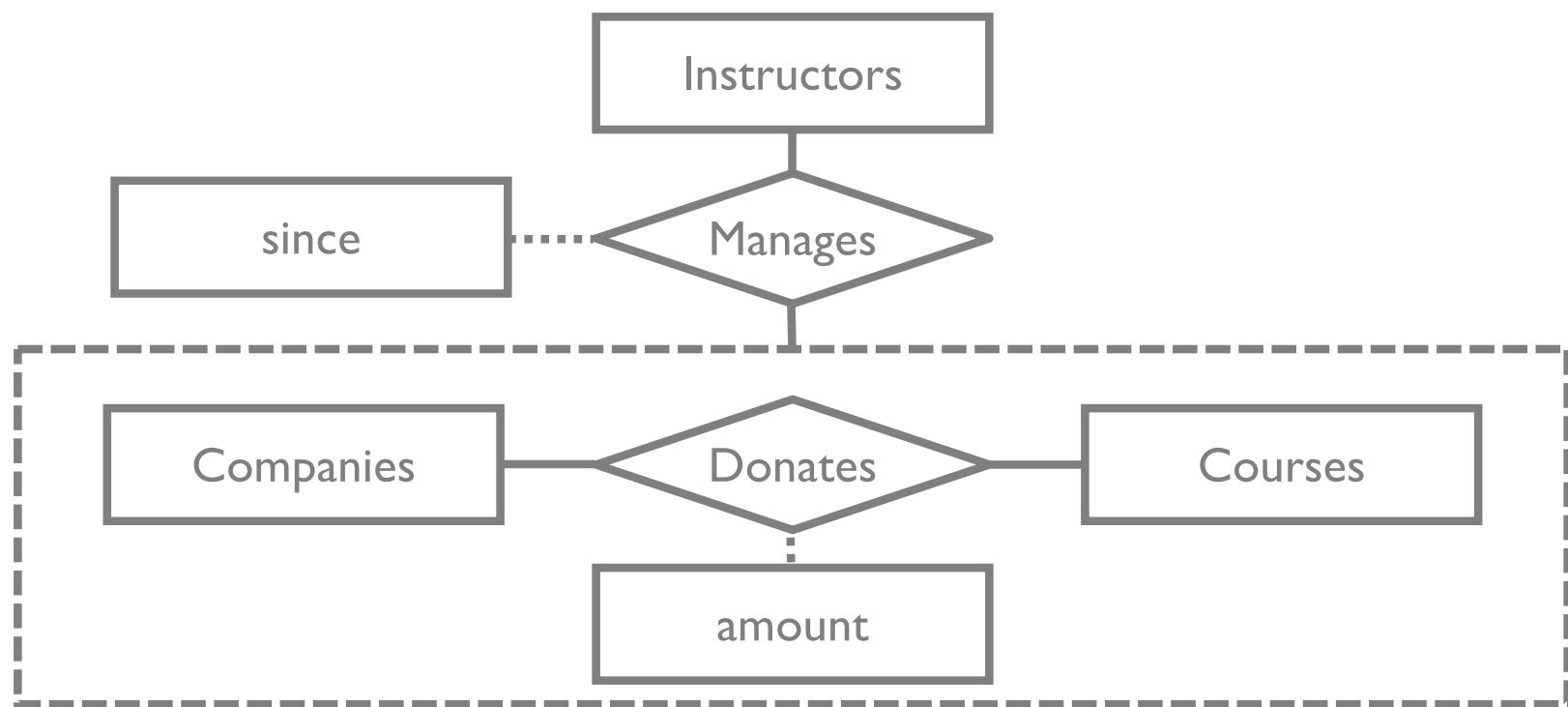
```
a int PRIMARY KEY,  
name text,  
b int,  
c int,  
CHECK (b is null or c is null) and  
(b is not null or c is not null)
```

)

Supports overlap = no, total = yes

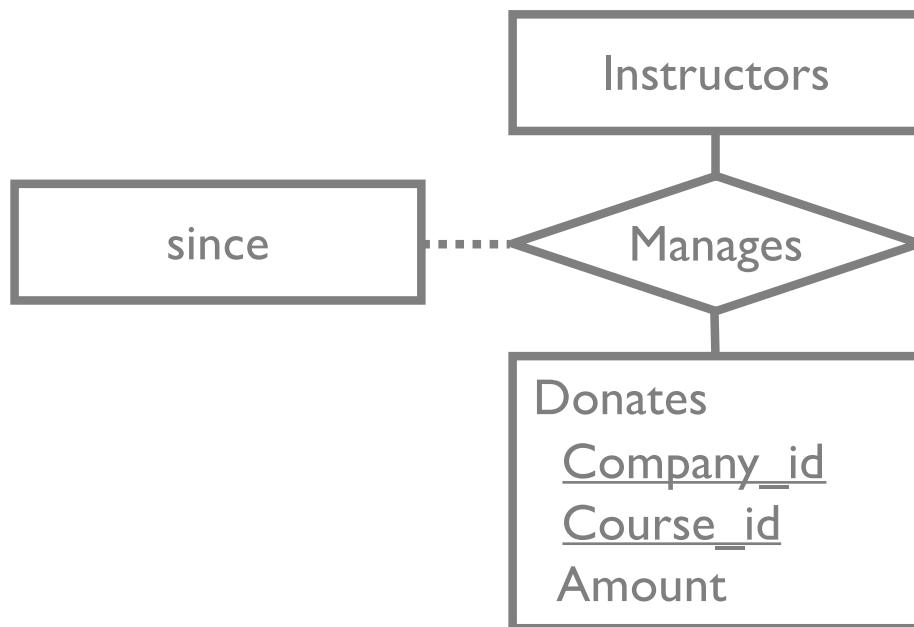


Aggregation



Aggregation

However you translate the donates relationship set is the table you use to model the relationship with instructors.

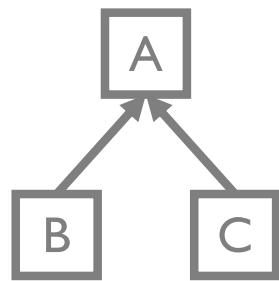


w4lll.github.io/quiz.html

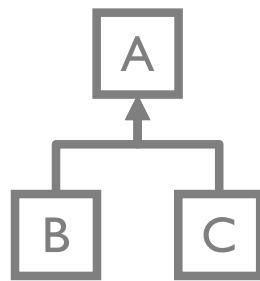
A(a int PRIMARY KEY, name text)

B(b int PRIMARY KEY, a int REFERENCES A)

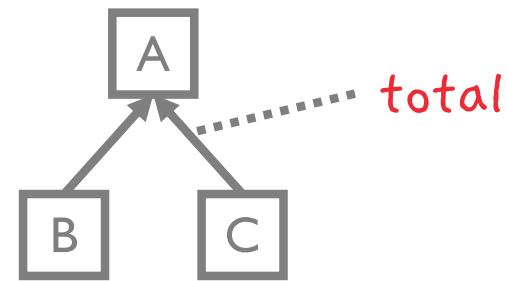
C(c int PRIMARY KEY, a int REFERENCES A)



(a)



(b)



(c)

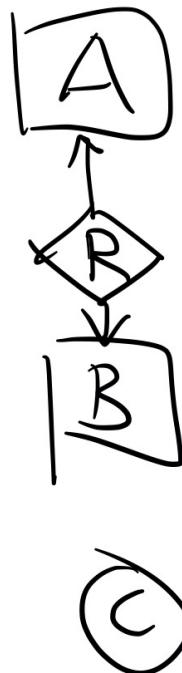
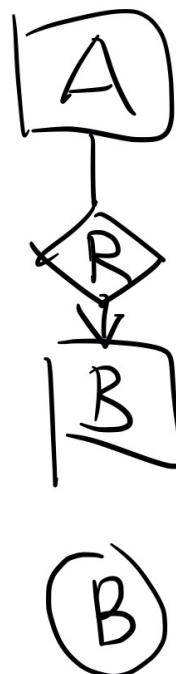
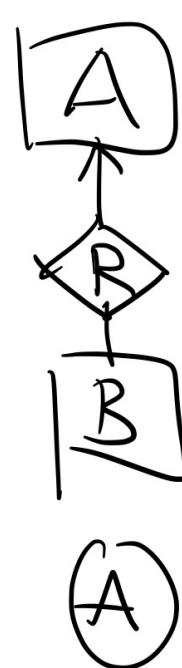
total

w4lll.github.io/quiz.html

```
CREATE TABLE R(  
    aid int primary key,  
    bid int references B(bid)  
)
```

a1
a2
(a1,b1),(a2,b1)
(a1,b2) // NO!

b1
b2



4 III In the Wild



Eirik Bakke @eirikbakke · 13h

Picking primary keys is hard.

...

0:13 AM ET and 6:30 PM ET on February 15 were not generated, but proposal su

in DUNS number to the New System for Award Management (SAM) Unique Entity

Research.gov. Choose the Login.gov option from sign in screen, enter Login.gov



Greetings! We now have a cloud offering

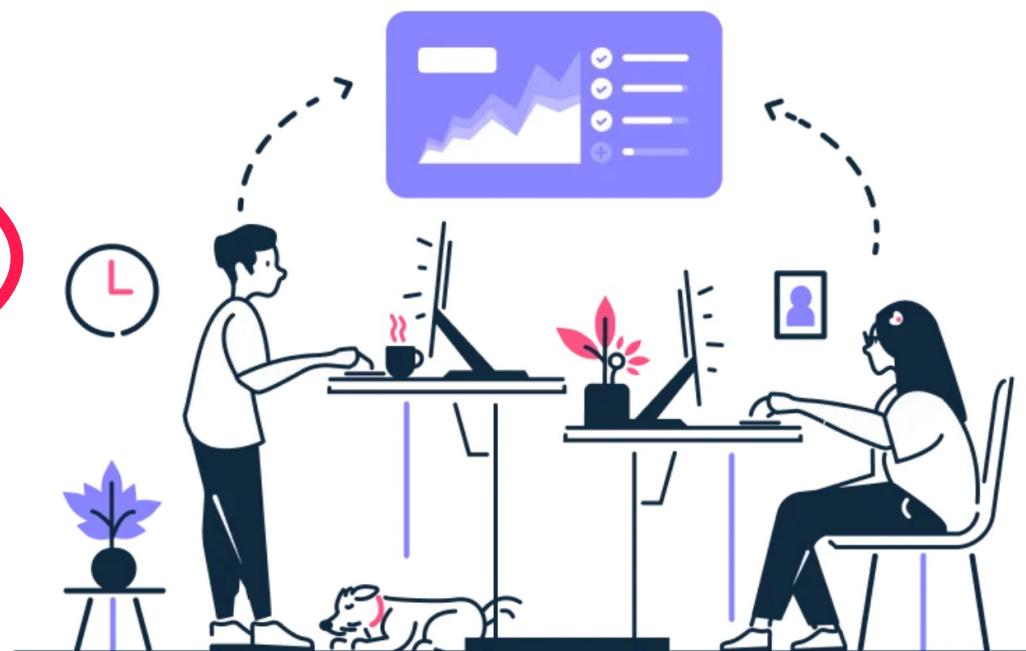
[Learn More](#)

Welcome to Great Expectations

Always know what to expect from your data

Great Expectations is a shared, open standard for data quality. It helps data teams eliminate pipeline debt, through data testing, documentation, and profiling.

JOIN OUR SLACK!



What is an Expectation?

An Expectation is a statement describing a verifiable property of data. Like assertions in traditional Python unit tests, Expectations provide a flexible, declarative language for describing expected behavior. Unlike traditional unit tests, Great Expectations applies Expectations to data instead of code.

Great Expectations' built-in library includes more than 50 core Expectations, such as:

- `expect_column_values_to_not_be_null`
- `expect_column_values_to_match_regex`
- `expect_column_values_to_be_unique`
- `expect_column_values_to_match_strftime_format`
- `expect_table_row_count_to_be_between`
- `expect_column_median_to_be_between`

Oh. These are integrity constraints

Recap

Relational Data Model Recap

Relation: schema, contains set of tuples

Schema: list of attribute, type pairs

Keys and integrity constraints

Foreign-key References

Schema

sid	name	age	gpa
1	eugene	20	2.5
2	luis	20	3.5
3	ken	33	3.9

Tuples

Cell

Other Popular Data Models

Matrix

TYPE is a field (+, * obey “usual algebra”)

Rows and Cols are ordered and indexed, not named

Values can be missing.

Lots of missing values is “sparse”. Else dense

TYPE	1	...	n
1			
...			
m			

A curved arrow points from the word "Cell" to the bottom-right cell of the matrix, which contains the letter "n".

Cell

Other Popular Data Models

Matrix: expressible as relation?

```
Matrix(  
    i int, j int,  
    v TYPE,  
    primary key (i, j)  
)
```

TYPE	1	...	n
1			
...			
m			

Other Popular Data Models

Relation expressible as matrix?

- ML expects matrices.
- Turn attribute types into field (e.g., a number)
- Hot-one encoding, featurization

<u>sid</u>	name	age	gpa
1	eugene	20	2.5
2	luis	20	3.5
3	ken	33	3.9



Other Popular Data Models

Value

TYPE	1
1	

Vector

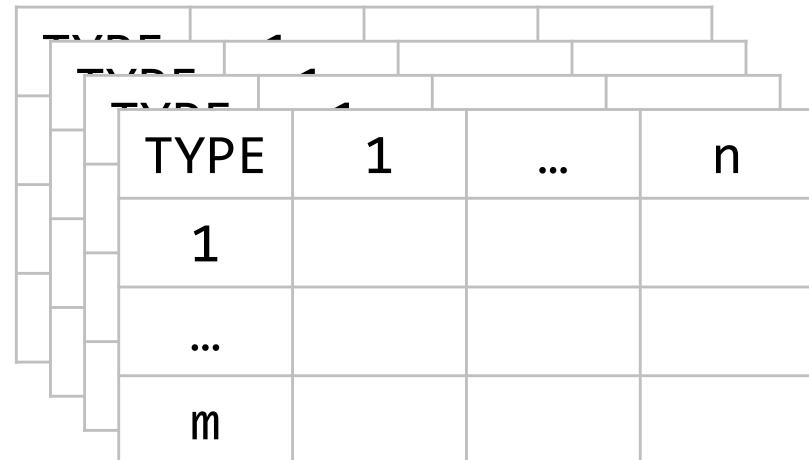
TYPE	1	2	...	n
1				

Matrix

TYPE	1	...	n
1			
...			
m			

Tensor

TYPE	1	...	n
1			
...			
m			



Other Popular Data Models

Pandas/R data frame

only columns have types

rows and cols are ordered and named

less mathematically grounded

	name ₁	name ₂	...	name _n
	type ₁	type ₂	...	type _n
name ₁				
...				
name _m				

REVIEW OF ER AND RELATIONAL

ER Overview

Ternary relationships

Relationships constraints

At most one

At least one

Exactly one

Weak entities

Aggregation

Is-A

Relational Review

Relations ~= Sets

Schema = structure // like a class definition

Instance = the data // like an object

"every relation is guaranteed to have a key"?

Integrity Constraints

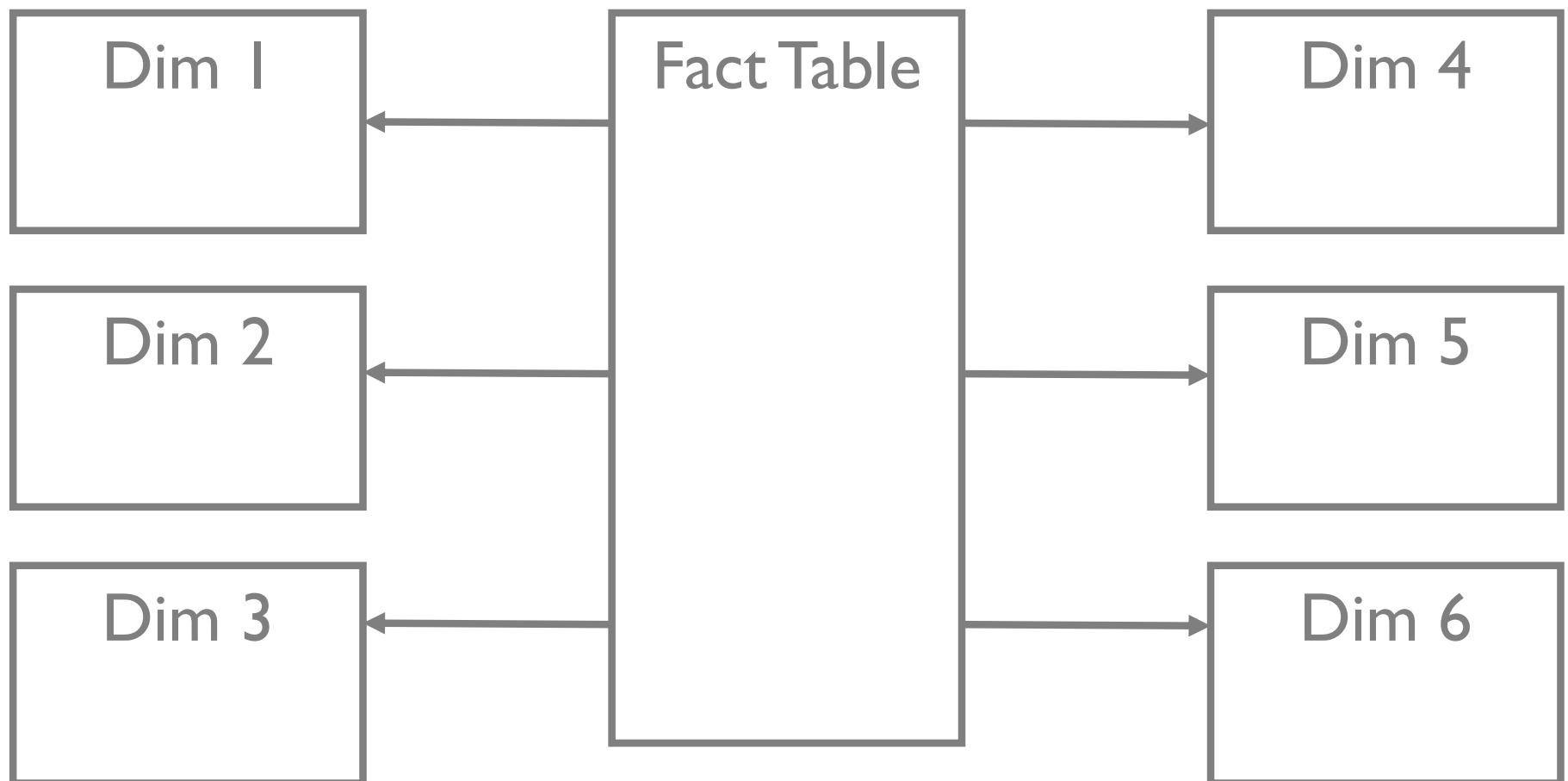
Candidate and primary keys

NOT NULL

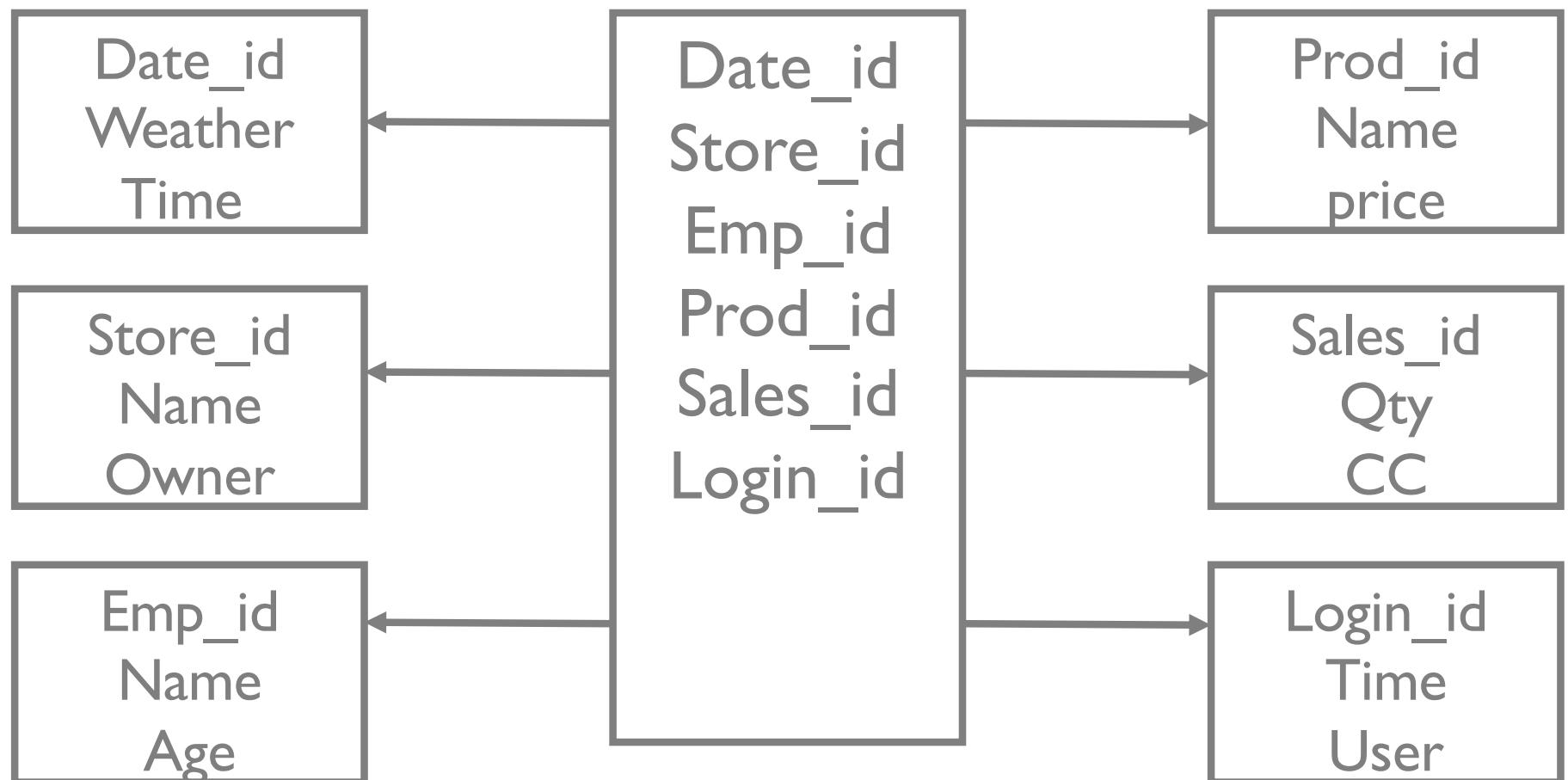
Referential integrity

How are foreign keys managed by the DBMS?

Star Schema



Star Schema



So What Happened?

1970 heated debates about CODASYL vs Relational Network arguments

- low level languages more efficient (performance)

- relational queries would never be fast (performance)

Relational arguments

- data independence

- high level *simpler* languages

Market spoke.

Other models beyond relational!

Summary

Better than IMS/CODASYL

- allows us to talk about constraints!

- allows us to talk at a logical level

- declarative queries better than navigational programs

Everything is a relation (table)

DBA specifies ICs based on app, DBMS enforces

- Primary and Foreign Keys most used

- Types == Domain constraints

SQL

Next Time

Relational Algebra

A set-oriented theory for relational data

Finish history lesson